HUMAN FOLLOWING WITH ENVIRONMENTAL SENSING AND MAPPING

by

Süleyman Batuhan Vatan

B.S., Mechanical Engineering, Istanbul Bilgi University, 2020

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Systems and Control Engineering

Boğaziçi University

2025

ACKNOWLEDGEMENTS

Preeminently, I wish to express my deepest gratitude to my supervisor, Prof. H. Işıl Bozma, for believing in me and welcoming me into her remarkable laboratory. Her hard work and unwavering dedication have continually inspired me, and her patient support has guided me throughout this research.

As for my thesis committee members, I would like to thank Prof. Yağmur Denizhan and Prof. Hakan Temeltaş for their valuable contributions.

I would like to thank my colleagues for cultivating a warm and supportive laboratory atmosphere. I am especially grateful to Serhat İscan for his guidance and to Arif Yılmaz, Alperen Sal, Ahmet Harun Eker, and Semanur Kumaş for their camaraderie. My heartfelt thanks also go to Yunus As for his steadfast support and inspiring suggestions. Working in this remarkable environment has been a privilege, and I have striven to be a "human dynamo," sustaining the spirit we have built together.

I am equally indebted to my friends, particularly Talha Keskin, whose invaluable insights have continually bolstered me. I thank Berkan Türkyılmaz, Bora Üzeyir İmamoğlu, Çiğdem Öztürk, and Emir Üstündağ for their unwavering confidence, and Servet Timuçin Yalçınkaya for first sparking my passion for engineering.

Finally, I extend my deepest appreciation to my family. My mother, sister, and grandmother have been pillars of strength, offering boundless encouragement. My father, together with my uncles Adnan and Salih and my aunt Ayşegül, has provided invaluable professional guidance, enriching me with wisdom and experience.

This work has been partially supported by TUBITAK EEEAG grant 119E203, TÜBİTAK TEYDEB 3230750, and ROYAL CB SBB 2019K12-149250.

ABSTRACT

HUMAN FOLLOWING WITH ENVIRONMENTAL SENSING AND MAPPING

This thesis is concerned with human following by a mobile robot that also needs to navigate in a socially compliant manner. The importance of this problem is due to the increasing usage of service robots in human-populated areas. This is a challenging task because the robot needs to consider proxemics while navigating. At the same time, it needs to keep the human in sight. In this thesis, this problem is studied from four aspects. First, an extensive evaluation of the previously proposed social navigation method, Social APF-RL, has been conducted to verify that the robot's navigation is socially compliant. Following, in order to better keep the human target within its field of view, a social robot head design has been realized based on a pan-tilt mechanism. Thirdly, human following in a socially compliant manner has been considered using the developed robot head, including body-head coordination while navigating. For this, a deep learning network based on reinforcement learning has been developed. Finally, recovery in case of human target loss is considered through human tracking and environmental sensing of doors using a specially trained YOLOv8 network. All the proposed methods are tested extensively in the Gazebo simulation environment.

ÖZET

ÇEVRESEL ALGILAMA VE HARİTALAMA İLE İNSAN TAKİBİ

Bu tez, sosyal olarak uyumlu yöngüdüm sırasında insanın takip etmesiyle ilgilidir. Hedef insan robot tarafından görülebiliyor ise, açık ortamlarda, bu görev oldukça basittir. Bununla birlikte, karmaşık ve insanlarla dolu ortamlarda, takip edilen insan ortamdaki diğer insanlar nedeniyle kısmen veya tamamen gizlenebilir veya engeller nedeniyle kaybolabilir. Dolayısıyla, sürekli olarak insan takibi yapabilmek adına, kaybolan hedefi tekrar bulabileceğimiz güvenilir bir yapıya ihtiyaç duyulmaktadır. Bu tezde, bu problem üç açıdan incelenmiştir. İlk olarak, mobil robot platformdan bağımsız olarak insan hedefini görüş alanında tutmak amacıyla, başka bir serbestlik derecesi getiren, bir kafa mekanizması geliştirilmiştir. Ardından, laboratuvarımızda daha önce geliştirilen sosyal yöngüdüm yönteminin yeterliği test edilmiştir. Bu testler kapsamlı bir şekilde hedef odaklı yöngüdümü inceliyor. Sonrasında, geliştirilen kafa mekanizması kullanılarak, yöngüdüm modelinin insan takibindeki yeterliliğiyle devam ediyor. Bulgularımıza dayanarak, vücut ve kafa mekanizması arasında koordinasyona ihtiyacımız olduğu sonucuna vardık. Ayrıca, insan hedefini kurtarmak için, duyusal bilgilere dayalı verileri engellere eşleyip, bu bilgiyi saklamak ve hangi engelin insan hedefinin kaybedilmesine yol açtığını anlamlandırmalıyız. Dolayısıyla hangi engelden sebep olduğunu bildiğimiz bu hedef kaybı, aynı şekilde tekrar bulunmasına imkan verecektir. Bu anlamda esas olarak engeller kapılar olarak belirlenmiştir.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS iii
ABSTRACT iv
ÖZET v
LIST OF FIGURES x
LIST OF TABLES xiv
LIST OF SYMBOLS xvi
LIST OF ACRONYMS/ABBREVIATIONS xxii
1. INTRODUCTION 1
1.1. Problem Statement 1
1.1.1. Social Navigation 1
1.1.2. Head Design for a Social Robot 2
1.1.3. Human Following via Body-Head Coordination 2
1.1.4. Tracking and Environmental Sensing 3
1.2. Contributions 3
1.3. Outline 3
2. SOCIAL NAVIGATION 5
2.1. Related Work 6
2.1.1. Human Detection 6
2.1.2. Social Navigation 7
2.2. Experimental Results 7
2.2.1. Simulation Tests 9
2.2.2. Real-Life Tests 11
3. SOCIAL ROBOT HEAD DESIGN 15
3.1. Related Work 15
3.2. Pan-Tilt Mechanism 18
3.3. Head Electronic Design 19
3.4. Head Control 22
3.5. Head Control Software 25

3.6. Robot Face . . . 25
3.7. Head and Robot Body Integration . . . 26
4. HUMAN FOLLOWING AND BODY-HEAD COORDINATION . . . 31
4.1. Related Work . . . 32
4.1.1. Deep Learning Based Human Following . . . 32
4.1.2. Human Following with Pan-Tilt Head Mechanisms . . . 33
4.2. Social APF-RL for Human Following . . . 35
4.2.1. Social APF-RL for Human Following: Theoretical Revisions . . . 35
4.2.2. Social APF-RL for Human Following: Experimental Evaluation 36
4.3. Head-Coordinated Social APF-RL for Human Following . . . 38
4.3.1. Action Set . . . 39
4.3.2. Observation Space . . . 40
4.3.3. Reward Function . . . 43
4.4. Environmental Complexity and Learning . . . 51
4.5. Simulation Setup for Training . . . 53
4.6. Learning Body-Head Coordination . . . 56
4.7. Experimental Results . . . 59
5. ENVIRONMENTAL SENSING FOR RECOVERY FROM TARGET LOSS 64
5.1. Related Work . . . 65
5.1.1. Recovery from Human Target Loss . . . 65
5.1.2. Obstruction Detection and Doors . . . 66
5.2. Tracking Objects of Interest . . . 68
5.3. Door Detection . . . 70
5.3.1. Learning Datasets . . . 70
5.3.2. Door Detection Learning . . . 71
5.4. Detecting Door Status and Orientation . . . 73
5.5. Target Loss and Door Finding . . . 76
5.6. Experimental Results . . . 77
5.6.1. Door Detection Performance . . . 77
5.6.2. Door Status Detection Performance . . . 79
6. CONCLUSION . . . 83

REFERENCES 86
APPENDIX A: OBSTACLE GENERATION AND MODELLING 96
A.1. Discontinuity-Based Clustering 96
A.2. Iterative End-Pigorithm Fit (IEPF) Algorithm 97
A.3. Ellipsoid Fitting (Khachiyan Method) 99
A.4. Comparative Obstacle Generation 101
APPENDIX B: SOCIAL APF-RL BASED NAVIGATION 102
B.1. Artificial Potential Functions 102
B.2. Admissibility of APF 103
B.3. Gradient Computation 104
B.4. APF-RL 106
B.5. Social APF-RL 108
APPENDIX C: SOCIAL FORCE MODEL (SFM) 110
C.1. Social Force Model (SFM) Force Terms 110
APPENDIX D: DOOR DETECTOR EVALUATION 113
APPENDIX E: SIMULATION'S USAGE 118
E.1. Environment Setup and Dependencies 118
E.1.1. System Requirements 118
E.1.2. Key Libraries and Tools 118
E.1.3. Building the Workspace 119
E.1.4. Sourcing Setup 119
E.2. Coordination and APF-RL Learning / Testing 119
E.2.1. Coord APF-RL Learning Setup 119
E.2.2. Using APF-RL Navigation 120
E.2.3. Head Tracking 120
E.2.4. Head and APF-RL Simultaneous Usage 121
E.2.5. Human Following with Head Tracking 121
E.3. Door Detection and Segmentation 122
E.3.1. Door Plugin Test 122
E.3.2. Door Detector and Status Packages 122
E.4. Free-Viewing Code 123

APPENDIX F: SIMULATION IMPROVEMENTS 124
APPENDIX G: SIMULATION PLUGINS 126
G.1. Model Data Plugin 126
G.1.1. Plugin Parameters 126
G.1.2. Plugin Custom Messages 126
G.1.3. Plugin Topics 127
G.1.4. Plugin Usage 127
G.2. RL Actor Control Plugin 128
G.2.1. Plugin Parameters 128
G.2.2. Plugin Custom Service Definitions 129
G.2.3. Plugin Services 129
G.2.4. Plugin Usage 130
G.3. RL Actor SFM Plugin 130
G.3.1. Plugin Parameters 131
G.3.2. Plugin Custom Service and Message Definitions 132
G.3.3. Plugin Services and Topics 132
G.3.4. Plugin Usage 133
G.4. Move Model Plugin 133
G.4.1. Plugin Parameters 133
G.4.2. Plugin Usage 133
G.5. Dynamic Gazebo Models Plugin 133
G.5.1. Plugin Parameters 134
G.5.2. Example Model Definition 134
G.5.3. Dynamics Manager 135

LIST OF FIGURES

Figure 2.1. AWS small house plan [17]. 9

Figure 2.2. Social navigation simulation tests: one room case 10

Figure 2.3. Social navigation simulation tests: Two rooms case. 11

Figure 2.4. Real world social navigation experiments without head mechanism 13

Figure 3.1. SempRob head design at the onset of the thesis [4]: (a) head design, (b) mechanical realization 16

Figure 3.2. SempRob head design. 19

Figure 3.3. Head attachment fixture, (a) isometric view (b) side-view. 20

Figure 3.4. (a) Pan tilt mechanism motors and (b) motor driver shield. 21

Figure 3.5. (a) Sparkfun serial breakout CH340C, and (b) XL4016 DC-DC buck converter. 23

Figure 3.6. Electrical diagram and communications of the SempRob. 27

Figure 3.7. Flow of processing for head control. 29

Figure 3.8. Robot face simulation [36]. 29

Figure 3.9. SempRob head and neck assembled. 30

Figure 3.10. SempRob, (a) head attachment and head rest, (b) final assembly with head attached. 30

Figure 4.1. Simulation corridor environment. 37

Figure 4.2. Human following in a corridor 38

Figure 4.3. Performance of Social APF-RL for human following in a corridor. 38

Figure 4.4. Body-head coordination model training and deployment pipeline. . 40

Figure 4.5. Reward function, $r_{fg}(a''(t), \omega''(t))$, (a) base reward, (b) extra reward based on $\dot{\delta}_{fg}$. 45

Figure 4.6. Reward function, $r_{fw}(a''(t), \omega''(t))$. 46

Figure 4.7. Follow window visualization and SempRob following the target. . . 46

Figure 4.8. Reward function, $r'_o(a''(t), \omega''(t))$. 47

Figure 4.9. Reward function, $r_h(a''(t), \omega''(t))$. 48

Figure 4.10. Reward function, $r_{\varphi_H}(a''(t), \omega''(t))$. 49

Figure 4.11. Reward function, $r_{h\alpha}(a''(t), \omega''(t))$. 50

Figure 4.12. Reward function, $r_{f_1}(a''(t), \omega''(t))$. 51

Figure 4.13. DRL and Gazebo plugin interaction pipeline. 54

Figure 4.14. Training environments 56

Figure 4.15. Episode rewards over 950000 iterations. 58

Figure 4.16. Body-head coordination test environments, (a) first trajectories, (b) second trajectories for each room. 59

Figure 5.1. Door detection training 72

Figure 5.2. Door status detection pipeline. 74

Figure 5.3. (a) Trained YOLOv8 door detector output, (b) rViz, depth points of robot camera looking through an open door. 79

Figure 5.4. Different door mesh models exported to Gazebo. 79

Figure 5.5. Door orientation detection simulation environments 81

Figure 5.6. Comparative door orientation detection errors 81

Figure 5.7. Comparative door orientation detection on door dataset: robot pose (a) looking at outwardly opening door, (b) directly looking through the door. 82

Figure A.1. Comparison of obstacle generation methods in AWS Small House [17]: (a) old obstacle generator, (b) revised obstacle generator. . . 100

Figure D.1. Door evaluation spider charts for in-house oriented door test image set: (a) YOLOv4, (b) YOLOv8 results. 115

Figure D.2. Door evaluation spider charts for in-house Human following test image set: (a) YOLOv4, (b) YOLOv8 results. 116

Figure D.3. Door evaluation spider charts for DWSonder test image set: (a) YOLOv4, (b) YOLOv8 results. 117

LIST OF TABLES

Table 2.1. Social navigation tests, one actor, goal oriented: small house simulation, one room. 10

Table 2.2. Statistical performance results: small-house simulation 12

Table 2.3. Statistical performance results: social navigation in real-world . . . 14

Table 3.1. Components of the head. 16

Table 3.2. Specifications of ROBOTIS Dynamixel MX-64T. 22

Table 3.3. SempRob electrical diagram component list. 24

Table 4.1. Body-head coordination actions and their ranges. 41

Table 4.2. Environmental complexities of training environments 57

Table 4.3. Test room environmental complexities. 61

Table 4.4. Comparative performance results: Social APF-RL Navigation, Social APF-RL for Human Following and Head-Coordinated Social APF-RL for Human Following. 63

Table 5.1. List of door image datasets. 71

Table 5.2. Door detector evaluation: aggregated results, top 5 weights by mAP. 78

Table 5.3. Door orientation detection: simulation data statistical results. . . . 81

Table 5.4. Door orientation detection: real-life data statistical results. 81

Table D.1. YOLOv8 door detector training arguments for each configuration. 114

Table D.2. Top 5 YOLOv4 weights sorted according to mAP50:95 on the oriented doors image set. 114

Table D.3. Top 5 YOLOv8 weights sorted according to mAP50:95 on the oriented doors image set. 114

Table D.4. Top 5 YOLOv4 weights sorted according to mAP50:95 on the human following image set. 114

Table D.5. Top 5 YOLOv8 weights sorted according to mAP50:95 on the human following image set. 115

Table D.6. Top 5 YOLOv4 weights sorted according to mAP50:95 result on the DWSonder image set. 115

Table D.7. Top 5 YOLOv8 weights sorted according to mAP50:95 result on the DWSonder image set. 116

LIST OF SYMBOLS

a	Action vectors for the DRL networks
A_{KF}	State-transition matrix of the constant-velocity model
b_1	Major axis of the elliptic obstacle
b_2	Minor axis of the elliptic obstacle
b_{1_0}	Initial major axis of the elliptic obstacle
b_{2_0}	Initial minor axis of the elliptic obstacle
b_{o_i}	Outer or internal definition of the ith obstacle for the repulsive field
c	Center of the robot
C_d	Dynamic complexity of the environment
C_s	Static complexity of the environment
c_x	Camera intrinsic parameter, x-coordinate of principal point
c_y	Camera intrinsic parameter, y-coordinate of principal point
$d(x)$	Stereo depth reading of the stereo camera at coordinate x $\in \mathbb{R}^2$
d_{RANSAC}	Perpendicular distance of RANSAC plane to the point at hand
D_{max}	Maximum depth threshold to mark visible human targets
$e_{f_1\alpha}(t)$	Yaw error between head pan and target human
$e_{r\alpha}(t)$	Yaw error between robot base and target human
f	PTU angles
f_1	Pan angle
f_2	Tilt angle
f_H^g	PTU human target angles
$f(x_h)$	Pan offset and tilt offset angle vector $f_1(x_h)$ and $f_2(x_h)$, respectively
f_x	Camera intrinsic parameter, focal length along the x-axis
f_y	Camera intrinsic parameter, focal length along the y-axis
g	Intermediate goal for the APF-based navigation
g^*	Final goal for the APF-based navigation

g_h	Target human goal for the APF-based navigation
h_b	Camera's base height
h_i	ith human
h_t	Camera's effective height, head tilt angle compensated
$\mathbf{H}$	Kalman Filter observation matrix
I_{pan}	Inertia of the head mechanism taken at pan axis
I_{tilt}	Inertia of the head mechanism taken at tilt axis
$\mathbf{I}$	Identity matrix of compatible dimension
k	Attractive potential field gain for the APF-based navigation
k_{β_i}	Repulsive potential field gain for the APF-based navigation
k_H	Attractive potential field gain for the APF-based PTU control
$\mathbf{K}_k$	Kalman gain at time step k
l_r	Length of the current room
$\hat{n}'_{door}$	Estimated normal of the door cloud using PCA
N	Number of beams in the down-sampled laser scan
N_1	Width of the image
N_2	Height of the image
N_h	Number of humans
N_o	Cardinality of $\mathcal{O}$
o_i	The ith obstacle position
P_1	Real world x coordinate of the stereo depth image
P_2	Real world y coordinate of the stereo depth image
P_3	Real world z coordinate of the stereo depth image
P_i	Real world coordinates of a stereo depth reading, $i \in \mathbb{R}^3$
P_o	Pivot point of the flip door
$\mathbf{P}$	State error covariance matrix
$\mathbf{P}_{k\|k-1}$	Predicted state error covariance at time step k
$\mathbf{P}_{k\|k}$	Updated state error covariance at time step k
q	Laser scan observation
q_h	Laser scan observation, human labels

q_{camera}	Binary laser scan observation that coincides with camera depth readings within 15m
$\mathbf{Q}$	Process noise covariance matrix
R	Quantitative reward gain for the DRL networks
R_{dh}	Penalty scalar applied in r_{dh}
R_{f_1}	Scaling factor for r_{f_1}
$R_{fg}, R_{fg}^{+}, R_{fg}^{-}$	Constant, positive and negative scalars used in r_{fg} respectively
R_{fw}	Scaling factor for r_{fw}
$R_{h\alpha}$	Scalar used in $r_{h\alpha}$
R_h	Scaling factor for r_h
R_{oi}	Scalars for the obstacle avoidance reward in r_o
$R_{overtime}$	Episode level penalty for overtime or target lost
R_{φ_H}	Scalar used in the head tracking reward
$R_{success}$	Episode level bonus for successful following
$\mathbf{R}$	Generic measurement-noise covariance matrix
$\mathbf{R}_{\text{RS-LIDAR-16}}$	Measurement noise covariance for the RS-LIDAR-16 scans
$\mathbf{R}_{\text{ZED2, 2m}}$	Measurement noise covariance for ZED2 stereo depth at 2 m
$\mathbf{R}_{\text{ZED2, 12m}}$	Measurement noise covariance for ZED2 stereo depth at 12 m
r	Total reward functions for the DRL networks
r_{dh}	Penalty function when the target human is not detected
r_{f_1}	Penalty function for keeping pan angle close to zero (discourages misalignment)
r_{fg}	Reward function for follow goal
r_{fw}	Reward function for keeping the robot inside the follow window
$r_{h\alpha}$	Reward function for aligning both base and head with the human
r_h	Social distance penalty function
r_o	Penalty for approaching or colliding with obstacles
r_{φ_H}	Reward function for head tracking error
r_{fg}	Reward function for follow goal

$T_{req,H}$	Torque requirement for PTU
u	Control effort for the APF-based navigation
u_{hr}	Vector from human to robot ($u_{rh} = -u_{hr}$)
u_{rh}	Vector from robot to human
v_1	x velocity of the tracked object in the planar odometry frame
v_2	y velocity of the tracked object in the planar odometry frame
v_h	Planar human velocity
v_i	The ith dynamic obstacle's velocity
v_r	Velocity of the robot
$v_{h_{target}}$	Planar velocity of the human target
v_{max}	Robot's maximum linear velocity
w_r	Width of the current room
x_1	x coordinate of the tracked object in the planar odometry frame
x_2	y coordinate of the tracked object in the planar odometry frame
x_h	Position of the target human in the RGB image
x_p	Horizontal pixel coordinate in the image
$\hat{x}^h_{odom}$	Planar position estimate of the target human in odometry frame
$\mathbf{x}$	Kalman filter state vector
$\hat{\mathbf{x}}_{k\|k-1}$	Predicted state at time step k given data up to $k-1$
$\hat{\mathbf{x}}_{k\|k}$	Corrected state at time step k after incorporating measurement
y_p	Vertical pixel coordinate in the image
$\mathbf{z}_k$	Measurement vector obtained at time step k
α_{comp}	Weighing for the internal and outer complexity terms
α_H	Angular acceleration of PTU's degree of freedoms
α_r	Heading of the robot
β	Repulsive potential field for the APF-based navigation

$\beta_{comp}^{internal}$	Complexity factor contributed by the pair-wise proximity of the internal obstacles
$\beta_{ij}^{internal}$	Distance between ith elliptic obstacle to the jth elliptic obstacle
β_{comp}^{outer}	Complexity factor contributed by the each obstacle's distance to the outer obstacle's center
$\beta_{i,comp}^{outer}$	Distance approximation between ith elliptic obstacle and the outer obstacle's center
β_i	Repulsive potential field constructed by ith obstacle
χ_{dh}	Binary flag: 1 if target detected, 0 otherwise
δ	Goal proximity of the robot
δf_H^g	Squared angular error between $f(t)$ and $f^g(t)$
δ_{fg}	Distance from robot to target human
δf_1^L	Horizontal LIDAR resolution
Δ_1	Horizontal field of view for the PTU's configuration space.
Δ_2	Vertical field of view for the PTU's configuration space.
Δ_1^c	Horizontal field of view for the PTU's configuration space, 110° for ZED2.
Δ_2^c	Vertical field of view for the PTU's configuration space, 70° for ZED2.
η	Ratio of area covered by internal obstacles with respect to outer one
γ	Attractive potential field for the APF-based navigation
κ	Scaling factor for the environmental complexity
μ_h	Elliptic obstacle expansion factor
ω	Observation vectors for the DRL networks
$\omega_{1,max}$	Maximum pan velocity
$\omega_{2,max}$	Maximum tilt velocity
ω_{max}	Robot's maximum angular velocity
ω_H	PTU's angular velocity vector
ρ_r	Robot radius
ρ_{o_i}	Effective radius of the ith elliptic obstacle

Σ	Covariance matrix of the door cloud centered at the pivot point
τ_H^g	Dead zone threshold for head tracking error
$\tau_{f_1\alpha}$	Yaw alignment threshold for head and target human
τ_{fw}^-, τ_{fw}^+	Inner and outer radii of the follow window
τ_h	Social distance threshold to humans
τ_o	Obstacle safety zone distance threshold
τ_p	APF-RL goal reach distance threshold
$\tau_{r\alpha}$	Yaw alignment threshold for base and target human
$\theta_1(x)$	Horizontal angle between the optical axis and ray passing through pixel x
$\theta_2(x)$	Vertical angle between the optical axis and ray passing through pixel x
$\theta_{door,i}$	ith door's orientation
θ_h	Angle of target human with respect to robot in odometry frame
θ_{fw}	Angular deviation of robot from the target human heading, used to find if robot falls into the follow window($[+\frac{\pi}{3}^\circ, -\frac{\pi}{3}^\circ]$)
θ_i	Orientation of the ith elliptic obstacle
φ	Admissible potential function for the APF-based navigation
φ_h	Attractive potential field for the APF-based PTU control
ε	Epsilon greedy coefficient
$\hat{\varphi}$	Non-admissible potential function for the APF-based navigation
ζ_i	Plane equation parameters, $i \in \{1, 2, 3, 4\}$
$\mathcal{D}_P$	Door cloud points
$\mathcal{F}_1$	Camera field of view that coincidents with the LIDAR scan points
$\mathcal{O}$	Obstacles

LIST OF ACRONYMS/ABBREVIATIONS

2D	Two-Dimensional
3D	Three-Dimensional
A2C	Advantage Actor Critic
A3C	Asynchronous Advantage Actor Critic
ACML	Adaptive Monte-Carlo Localization Algorithm
AHP	Analytical Hierarchy Process
AP	Average Precision
APF	Artificial Potential Function
APF-RL	Artificial Potential Function Reinforcement Learning
AR	Average Recall
AWS	Amazon Web Services
CNN	Convolutional Neural Network
DC	Direct Current
DO1	Plane normal comparison
DO2	Comparison of door normal and door pivot axis
DRL	Deep Reinforcement Learning
DQN	Deep Q-Network
E2E	End-to-End
ECE	Expected Calibration Error
EKF	Extended Kalman Filter
ESFM	Extended Social Force Model
FOV	Field Of View
FPPI	False positives per image
GPIO	General Purpose Input/Output
GNN	Graph Neural Network
HAR	Human Activity Recognition
HOG	Histogram of Oriented Gradient
HSR	Human Support Robot

ID	Identification
IEPF	Iterative End-Point Fit Algorithm
IMU	Inertial Measurement Unit
IoU	Intersection over Union
KD-Tree	K-dimensional tree
LIDAR	Light Detection and Ranging
LR	Learning Rate
LSTM	Long-Short Term Memory
mAP50:95	Mean Average Precision Calculated at Intersection Over Union Thresholds (from 0.5 to 0.95)
PCA	Principal Component Analysis
PCL	Point Cloud Library
PDR	Pedestrian Dead Reckoning
PID	Proportional–integral–derivative
POMDP	Partially Observable Markov Decision Process
PTU	Pan-Tilt Unit
RANSAC	Random Sample Consensus
RGB	Red, Green, and Blue
RGB-D	Red, Green, and Blue with Depth
RL	Reinforcement Learning
ROI	Region of Interest
ROS	Robot Operating System
RMSprop	Root Mean Squared Propagation
SAC	Soft-Actor Critic
SARL	Socially Attentive Reinforcement Learning
SARSA	State-Action-Reward-State-Action
SDF	Spatial Data File
SFM	Social Force Model
SGD	Stochastic Gradient Descent
SIFT	Scale-Invariant Feature Transform
SLAM	Simultaneous Location and Mapping

SPT	Spencer People Tracking
SVM	Support Vector Machine
TTL	Transistor-Transistor Logic
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
VRAM	Video Random Access Memory
YOLO	You Only Look Once

1. INTRODUCTION

Social robots are rapidly evolving and increasingly participating in human-robot interactions across a variety of contexts [1]. These robots are being integrated into a wide range of applications, from healthcare to service industries, enabled by their ability to follow context-specific social norms. Following guiding humans through cluttered and human populated environments remains a significant challenge. This thesis addresses this problem and aims to develop a robot capable of socially compliant human following via head-body coordination.

1.1. Problem Statement

In this endevour, this thesis has studied the following four related problems:

- Social navigation
- Head design for a social robot
- Human following through body-head coordination
- Tracking a human subject and re-acquiring the target if it is temporarily lost

1.1.1. Social Navigation

Social robots are becoming an important factor in modern-day life; they appear in hospitals to assist nurses, care for the elderly, assist in services, and act as personal guides to direct humans to certain places. These environments require a fundamental functionality of the robot being able to handle human-robot interactions. Simply moving around in human-populated environments or following a specific human requires the robot to detect the humans and track their movement while safely and socially navigating in the environment. These movements have a set of rules that robots must comply with since the robot shares the same environment as humans. In this context, respecting personal space, namely proxemics, is a priority to ensure human comfort.

These elements must be accounted for social navigation.

1.1.2. Head Design for a Social Robot

Social robots that share the same environment as humans need to have physical appearances that are acceptable by the humans - as they interact with them. This is in contrast to industrial robots that do not need to have similar appearance features since they are evaluated based on their functionality. There are five criteria listed for designing social robots: form, modality, social norms, autonomy, and interactivity [1]. Hence, these need to be taken into account. The second problem addressed in this thesis is concerned with ‘formaspect. In particular, we aim to equip our robot with a head mechanism that not only enables it to perceive nearby humans but also provides it with a sympathetic facial expression. Since a head addition to the robot is a humanoid feature, we need to take the uncanny valley phenomenon into account. The condition of not being able to feel closeness to the robots is called “Uncanny Valley” [2, 3]. This is not desired since people have a eerie sensation of the robot and hence prefer not to interact with it.

1.1.3. Human Following via Body-Head Coordination

Adding a head mechanism to move the robot’s sensors introduces an extra degree of freedom, improving its ability to perceive and navigate. However, it also brings challenges — primarily the need for coordinated head-body movement to avoid socially inappropriate behavior. While the head helps maintain visual contact with the human target, the robot must also navigate around obstacles. Sudden body movements can disrupt head tracking, and maintaining sight of the target should not come at the cost of collisions. Effective decision-making requires the robot to consider environmental context and plan waypoints that balance target tracking and safe navigation.

1.1.4. Tracking and Environmental Sensing

The human target may become partially or fully occluded, often due to environmental structures like doors or corners, leading the robot to lose visual contact. To address this, a robust social navigation strategy is needed — one that maintains sight of the target when possible and enables recovery after temporary obstructions. In our approach, we focus on recognizing doors and assessing their status to better understand their role in target occlusion. By tracking both the human and doors, the robot can infer whether a loss of visual contact is due to a door and respond appropriately to resume following.

1.2. Contributions

The contributions of the thesis can be summarized as follows:

- Social navigation: Extensive tests for the previously developed social navigation method, then using the head mechanism along with it for the human following scenario,
- Pan-tilt head design: A socially plausible head mechanism design that can be integrated with the mobile robot body,
- Body-head coordination for effective use of the head mechanism in socially compliant navigation.
- Target loss recovery through tracking and door detection, along with its status.

1.3. Outline

The outline of the thesis is as follows:

- Chapter 2: Extensive evaluation of the social navigation method is presented.
- Chapter 3: The pan-tilt head design and realization are presented.
- Chapter 4: A novel approach to body-head coordination for human following is

introduced

- Chapter 5: Tracking and door detection approaches to recovery from target loss are presented.
- Chapter 6: A brief summary of the thesis and some suggestions regarding future work are discussed.

2. SOCIAL NAVIGATION

This chapter evaluates the previously developed Social APF-RL navigation method [4] - an extension of APF-RL navigation [4,5]. Unlike APF-RL navigation, Social APF-RL navigation takes the humans in the robot's surroundings into account and aims to keep its distance from them in accordance with proxemics. For completeness, Social APF-RL method is explained in detail in Appendix B.

The evaluation had two goals: First, a more comprehensive evaluation of Social APF-RL based navigation was needed, as it had not been tested thoroughly. Second, its suitability for usage in human following scenarios needed to be determined. The contributions of this chapter can be summarized as follows:

- Extensive tests have been conducted to quantify the performance of the previously developed Social APF-RL navigation method. This work has been presented in [6].

The outline of the chapter is as follows: Related literature is discussed in Section 2.1. The extensive evaluation of the Social APF-RL approach is presented in Section 2.2.

2.1. Related Work

Social navigation is a complex problem as it involves several subproblems:

- Human detection
- Navigation considering detected humans

The majority of research tackles these challenges separately, with relatively few studies studying both aspects simultaneously. Hence, in the rest of this section, related work addressing each subproblem is reviewed briefly.

2.1.1. Human Detection

Proposed human detection methods vary by data type—such as RGB, RGB-D, LIDAR, IMU, or combinations thereof—and approach, ranging from algorithmic to deep learning-based. For example, one method detects and tracks humans by masking the background with depth data, extracting Histogram of Oriented Gradient (HOG) features from the foreground RGB images, classifying them with Support Vector Machines (SVM), and using a Kalman filter for tracking [7]. Another approach combines RGB-D and LIDAR data with a Single Shot MultiBox Detector (SSD) to track people, claiming robustness to occlusion and illumination changes and enabling recovery when the target leaves the robot's camera field of view [8]. A hierarchical method updates the followed person's state using RGB-D and 2D LIDAR data, emphasizing leg tracking via geometric features to maintain natural following distance [9]. Similarly, [10] fuses upper-body detection from an RGB-D camera with leg detection from 2D laser scans, integrating data via nearest-neighbor matching and an Extended Kalman Filter (EKF) within the Spencer People Tracking framework. Another study employs Kinect's skeleton tracker combined with a color-based Camshift tracker; when stable, tracking relies on the skeleton and EKF, otherwise it switches to Camshift, initiating tracking from hand gestures [11]. Lastly, some systems track humans based on measurements from wearable IMU devices [12].

In parallel, deep learning based detector models are being increasingly used for human detection. A pre-trained MobileNet V2 model to detect human subjects along with the Intel Realsense camera proposes a classification-based human following in three parts: i) Initialization in which a ResNet-18 is used to train a classification model using RGB images as to identify the human target in the occluded scene; ii) Tracking as to differentiate the target from the detection candidates coming from the MobileNet V2; iii) Robot control that assesses the required actions the robot must take to follow the target [13]. The downside of this work is that the time it takes for the initialization phase to complete, thus starting the human follow, also takes time.

2.1.2. Social Navigation

The proposed approaches vary in their use of local planners and maps. For instance, the "Follow the Gap" local planner directs the robot toward the human target while avoiding obstacles, maintaining a user-defined following distance, with velocity commands regulated by a PID controller [10]. To ensure socially compliant navigation, the APF-RL method has been extended to Social APF-RL, which incorporates social norms by maintaining distance from humans' personal spaces [4, 5]. Other approaches leverage deep reinforcement learning, employing CNNs based on Deep Q-Network (DQN) and Asynchronous Advantage Actor Critic (A3C) algorithms for navigation in crowded settings [14]. The Extended Social Force Model (ESFM) simulates pedestrian behavior with added features for group interactions, such as street conversations and group entries. Furthermore, reinforcement learning methods combining DQN, adapted CNNs, and graph neural networks are used to extract environmental features and model complex interactions among objects, obstacles, and the robot [15].

2.2. Experimental Results

A comprehensive experimental evaluation of Social APF-RL navigation has been conducted. Although previous tests included both simulations and real-world trials, they were limited—simulations closely resembled the training environment, and real-

world testing involved only one scenario with 10 trials [4]. The new experiments address these limitations by expanding test conditions. Specifically:

- Simulation tests were extended to a more realistic household environment featuring non-convex objects. Additionally, a two-room scenario was introduced to assess whether the model's performance generalizes beyond the originally trained single-room setting.
- Real-world evaluation was broadened by testing the model across four different setups for a more thorough assessment.

In all the tests, human detection is obtained from a YOLO-based model along with the RGB-D (ZED2 stereo) camera's detector and tracker. Performance is studied considering the following criteria:

- Success rate - Percentage of reaching the goal
- Failures due to collisions
- Percentage of not reaching the goal
- Average time
- Average distance to human
- Minimum distance to human
- Ratio of total travel distance and shortest distance to goal

Our experimental evaluation also includes a comparative study in which we compare performance with respect to the classical navigation method APF, APF-RL, and the Social Force Model (SFM) [16] as baselines. Summaries of these methods are given in Appendix B.1, Appendix B.4, and Appendix C, respectively. Note that if we conclude that the model is appropriate to use in social navigation, we will use it for the human following scenarios.

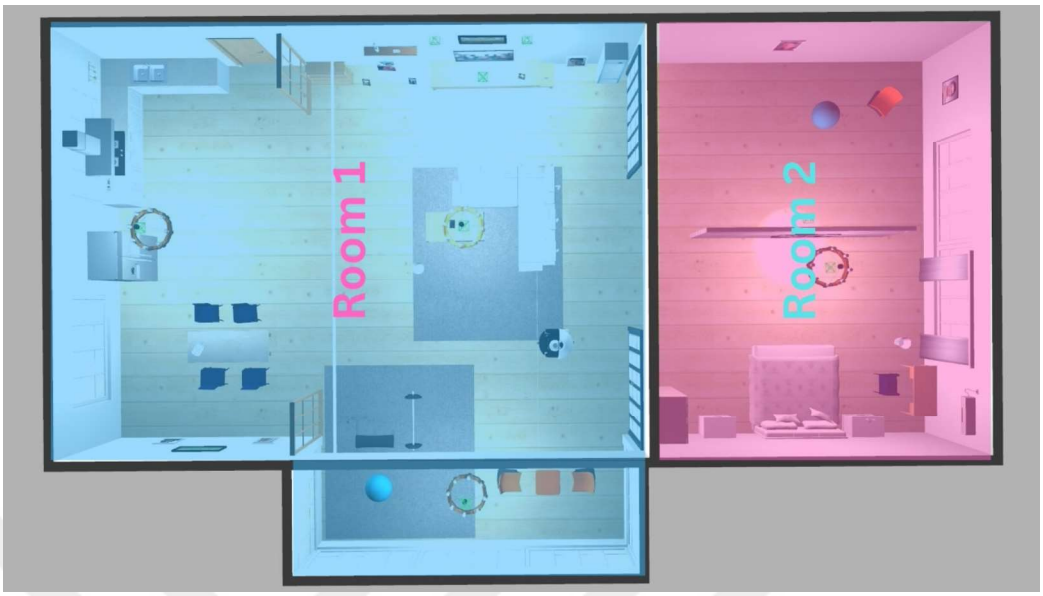


Figure 2.1. AWS small house plan [17].

2.2.1. Simulation Tests

For simulation test setups, Amazon Web Services (AWS) small house simulation environment has been used [17]. The plan of the AWS small house environment can be seen in Figure 2.1. As seen, it consists of two rooms connected through a sliding door. One-room tests are done in Room 1 as shown in Figure 2.1. In particular, an actor traverses the room diagonally while the robot is trying to reach a goal point in the same room, as shown in Figure 2.2. For each navigation method, there are 20 trials. Performance results are given in Table 2.1. It is observed that while Social APF-RL has performance similar to the other methods in regards to successful task completion, ensuring that the robot's average distance to humans is as far as possible.

The following two-room tests will be conducted on the AWS small house simulation environment. There are four cases as seen in Figure 2.3. The case 1 human movement is given in Figure 2.3(a), where it directly crosses through the door, making robot hard to traverse between rooms. Cases 2 and 4 are essentially the same, given in Figure 2.3(b) and Figure 2.3(d), respectively, but in case 4, we have two people,

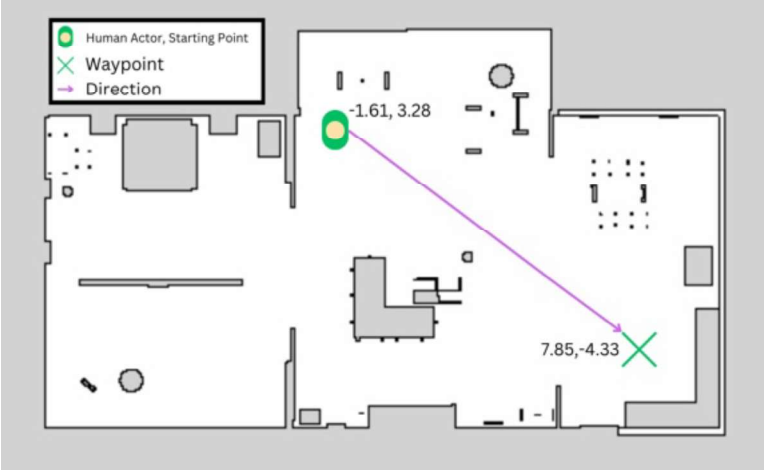


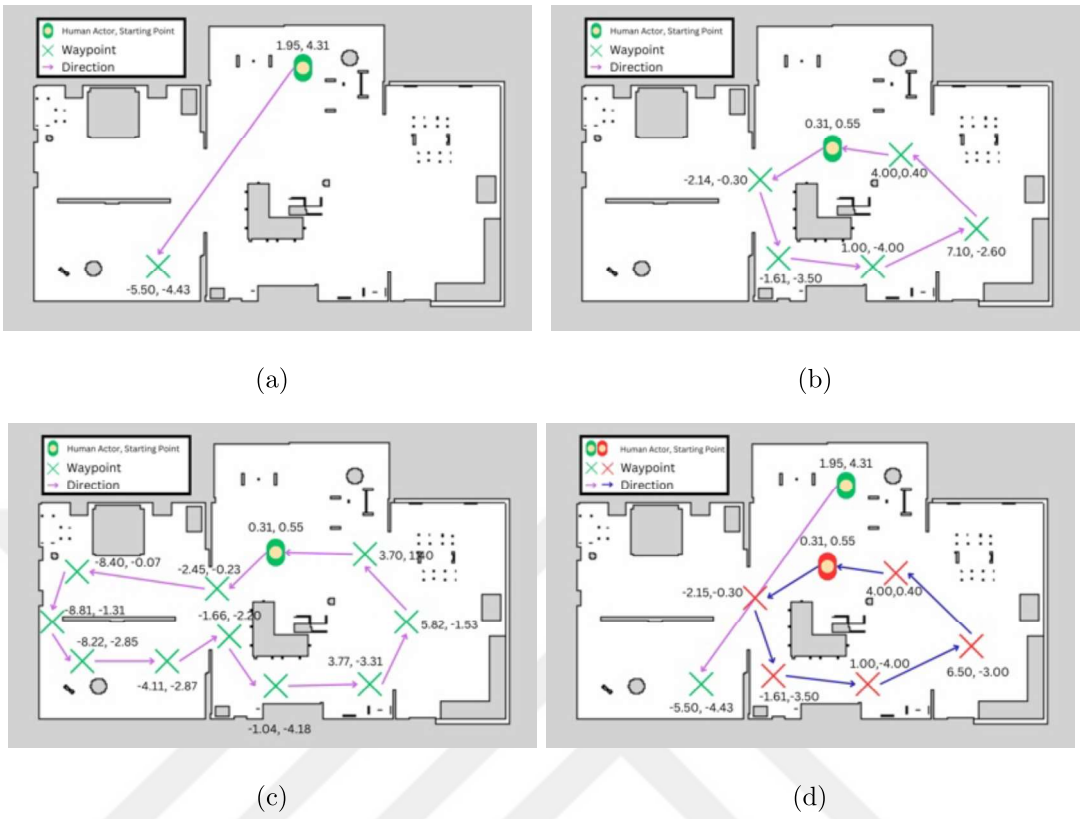
Figure 2.2. Social navigation simulation tests: one room test case without head mechanism.

Table 2.1. Social navigation tests, one actor, goal oriented: small house simulation, one room.

Method	Success Rate (%)	Failure Due Collision (%)	Stuck at Local Minima (%)	Avg. Elapsed Time (sec)	Avg. Dist. to Human (m)	Avg. Min. Dist. to Human (m)	Travel Dist (%)
APF	85%	10%	5%	20,5800	5,4879	2,4025	123,1476
APF-RL	85%	15%	0%	24,3919	5,3126	1,9831	133,3817
SFM	50%	50%	0%	11,2354	5,1312	3,6393	134,7215
Social APF-RL	80%	20%	0%	18,7039	5,6824	2,0040	122,4943

essentially mixing case 1 and 2. Case 3, on the other hand, human trajectory loops around the two rooms. Regardless of the case setup, the robot either spawns in Room 1 or Room 2 (see Figure 2.1) and has a goal in the opposite room. Each case has 18 trials for all navigation methods that we are comparing, a total of 288 runs. Note that in these test cases, the head mechanism was not utilized. The cases for two-room tests are described below:

(i) A human actor goes through the doorway with a straight path in a cyclic manner,
(ii) A human actor loops over the pre-defined trajectory, one waypoint being the middle of the doorway cyclically,
(iii) A human actor loops throughout two rooms in a cyclic manner,
(iv) Similar structure of both cases 1 and 2, but simultaneously.



(a) (b)

(c) (d)

Figure 2.3. Social navigation simulation tests: two rooms case without head mechanism: (a) Case 1 - through the door, (b) Case 2 - with one person, (c) Case 3 - looping through two rooms, (d) Case 4 - with two people.

Performance results for the two-room tests are given in the Table 2.2. Results indicate that even if the Social APF-RL achieves the highest success rates is not sufficient and we can not directly use our model as it is to traverse through multiple room environments, which are essential for human following.

2.2.2. Real-Life Tests

Real-life test scenarios for one room consist of four different cases. Each trial is repeated 15 times, for all cases and all navigation methods, a total of 240 runs. Those cases can be seen in Figure 2.4. Four different cases for real-life tests are as follows:

Table 2.2. Social navigation performance results for small house simulation (two rooms): one actor, goal oriented.

Method	Success Rate (%)	Failure Due Collision (%)	Stuck at Local Minima (%)	Avg. Elapsed Time (sec)	Avg. Dist. to Human (m)	Avg. Min. Dist. to Human (m)	Travel Dist (%)
APF	6%	86%	8%	14,2153	3,1958	1,5354	121,04%
APF-RL	21%	40%	39%	21,1288	4,9527	2,1676	123,24%
SFM	10%	75%	15%	15,8553	6,3237	1,7012	185,10%
Social APF-RL	39%	29%	32%	17,3482	5,4395	2,8312	167,75%

(i) No internal obstacle, just one person moving toward the robot can be seen in the Figure 2.4(a),

(ii) Cluster of objects in the middle, where a circular path around that cluster remains traversable, and a human walks towards the robot can be seen in the Figure 2.4(b),

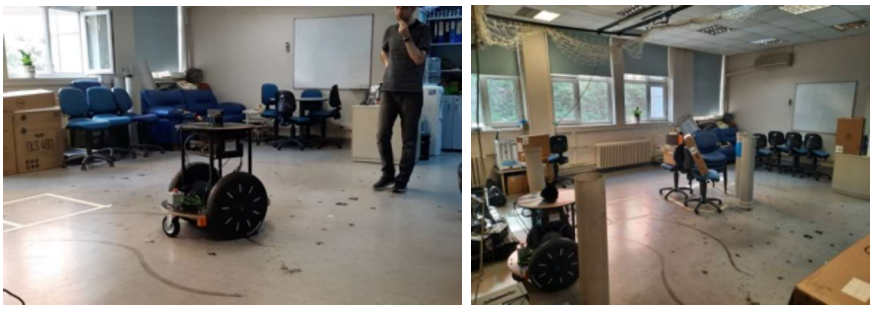
(iii) Double C-shaped obstacle formation, leaving a corridor that connects them, and a human walks through that corridor can be seen in the Figure 2.4(c),

(iv) Same obstacle formation as the second case, with two people traversing around the room can be seen in the Figure 2.4(d).

Results are given in Table 2.3. It is observed that overall, the Social APF-RL has the highest success rate while keeping the overall highest average human distance. We can conclude that both the simulation and the real-life results are similar in terms of quantitative findings. It is observed that the Social APF-RL is an adequate model for social navigation for one room. However, its reliability for navigation in multiple room environments needs to be improved. This is possibly attributed to two factors:

(i) As the learning model is trained on one-room scenarios, its performance in a multi-room navigation case is low.

(ii) While the obstacle generator model is sufficient for one-room navigation, it is not applicable for multi-room navigation scenarios.



(a) (b)

(c) (d)

Figure 2.4. Social navigation real tests, without head mechanism: (a) Case 1 - without obstacles, (b) Case 2 - with center obstacle, (c) Case 3 - local minima setup, (d) Case 4 - with two people.

Table 2.3. Statistical performance results for social navigation tests in one room.

Case #	Method	Success Rate (%)	Failure Due Collision (%)	Stuck at Local Minima (%)	Avg. Elapsed Time (sec)	Avg. Dist. to Human (m)	Avg. Min. Dist. to Human (m)	Travel Dist (%)
Case1	APF	100%	0%	0%	5,9833	3,0391	1,5685	113,68%
	APF-RL	93%	7%	0%	12,5352	3,2123	1,5138	160,87%
	SFM	100%	0%	0%	28,4474	3,1548	1,2298	203,28%
	Social APF-RL	93%	7%	0%	8,5185	3,5036	1,7073	122,25%
Case2	APF	60%	33%	7%	28,2922	1,3709	0,7705	152,02%
	APF-RL	93%	7%	0%	21,6370	2,1888	0,9063	158,39%
	SFM	53%	47%	0%	23,2554	2,2976	0,9010	168,77%
	Social APF-RL	100%	0%	0%	18,1395	2,3246	1,0648	138,70%
Case3	APF	0%	27%	73%	-	-	-	-%
	APF-RL	87%	7%	7%	35,0529	2,3300	1,0135	233,29%
	SFM	73%	27%	0%	28,4080	2,6577	0,9170	206,50%
	Social APF-RL	93%	7%	0%	28,4285	2,5967	1,0142	196,63%
Case4	APF	40%	20%	40%	35,6640	2,6373	2,1077	184,04%
	APF-RL	100%	0%	0%	52,3747	2,5145	0,7506	188,44%
	SFM	80%	20%	0%	30,5452	2,3220	0,7968	158,80%
	Social APF-RL	100%	0%	0%	61,6339	2,6605	0,8740	167,01%

3. SOCIAL ROBOT HEAD DESIGN

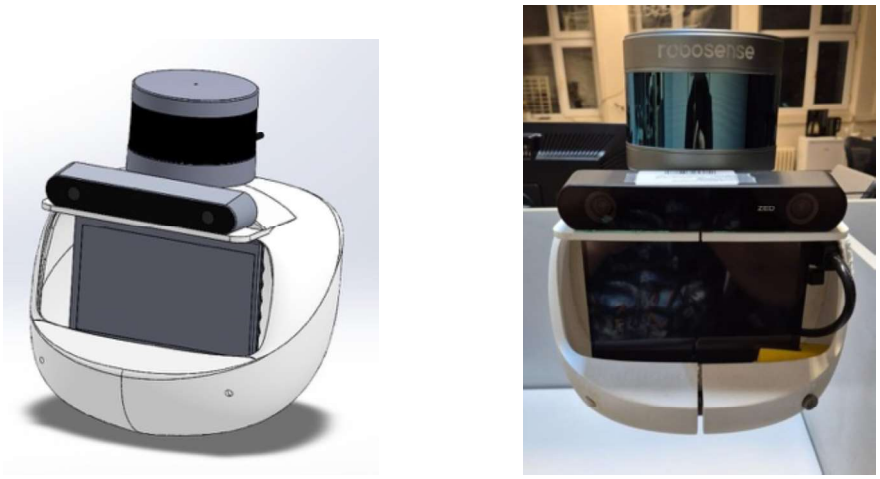
This chapter is focused on completing the robot's head that had been started in previous work [4]. A head design and its mechanical realization had been made as shown in Figure 3.1(a) and Figure 3.1(b), respectively with components as listed in Table 3.1. A stereo camera is located on top of the head mechanism as specified in Table 3.1, giving RGB and depth images. However, its completion and deployment were deferred to future work. This thesis addresses these tasks as follows:

- Revising mechanical design for improved sensor integration: Ensuring all necessary sensors (RGB-D camera and LIDAR) are incorporated into the head assembly.
- Designing the pan-tilt mechanism: Developing the mechanical, electromechanical, and electronic components to serve as a neck, enabling two degrees of freedom in head movement.
- Implementing velocity control and software for the pan-tilt mechanism: Using visual feedback to direct the head to track the object of interest.

Related literature is discussed in Section 3.1. The details the design and realization are presented in the rest of this chapter. Section 3.2 outlines work done to finish the head design. Namely, first, we discuss selecting appropriate electronic components in Section 3.3. Then, we discuss controlling the head in Section 3.4 and explain the respective software design and development in Section 3.5. For enhancing the social interaction, we discuss adding a face to the head mechanism in Section 3.6. Lastly, the head mechanism is attached to the mobile platform as detailed in Section 3.7.

3.1. Related Work

The head is a critical component in the design of social robots. This is partly attributed to the fact that a robot's face plays a key role in its social acceptance. Head



(a) (b)

Figure 3.1. SempRob head design at the onset of the thesis [4]: (a) head design, (b) mechanical realization

Table 3.1. Components of the head.

#	**Component**	**Weight (kg)**
1	Printed Head Casing	0.780
2	Waveshare 7” Capacitive Touch Screen LCD (H)	0.418
3	Robosense RS-Lidar 16	0.870
4	StereoLabs ZED 2 Stereo Camera	0.124
Total		2.192

design can either replicate human head mechanics or prioritize technical optimization without concern for human likeness—known as a technomorphic head [18]. The design must address both head motion and the robot’s facial representation.t.

Head motion in robots is typically implemented using pan-tilt unit (PTU) mechanisms that provide two degrees of freedom—pan and tilt [19–25]. These mechanisms serve to extend the limited field of view (FOV) of onboard cameras [26]. PTUs offer several advantages, such as enabling gaze control and stabilization [19], as well as allowing fast tracking of objects of interest to support active perception of the environment [24]. Without such a mechanism, a mobile robot must move its entire body

to maintain an object within its sensory range or to scan the environment for objects [20, 24]. From a control perspective, the camera must continuously center itself on the object of interest [19, 20, 23, 24, 26–28]. This problem has been widely studied. For instance, [19] proposes active parameter adjustment using a processing window to keep the target centered. In [24], features extracted from images are paired with two active disturbance rejection controllers. A PID controller, tuned via a backpropagation neural network, is used in [27] to address performance limitations arising from nonlinear servo characteristics. Similarly, dynamic models of the mobile robot and image features are leveraged in [23] to track targets, while model uncertainties are managed using a radius-based neural network. Additionally, object search strategies employ probabilistic frameworks—such as a Bayesian grid map in conjunction with SIFT features and predefined pan-tilt angles to scan the environment [20]. The use of PTUs in mobile robots extends beyond tracking. For example, they are employed to detect signs to aid in robot self-localization [29]. PTUs are also integrated into gaze control systems that support simultaneous localization and mapping (SLAM) and path planning to enhance navigation in unknown indoor environments [22].

A head design for a robot typically also requires the inclusion of a face. One of the earliest notable examples of a robot incorporating both a head and a screen-based face is the Roboceptionist, developed in the early to mid-2000s [30]. The head mechanism not only supports sensor movement but also provides a platform for social interaction through facial expressions—an essential aspect of human social communication [18, 31, 32]. These non-verbal cues allow the robot to communicate its internal state [31], thereby making it more relatable and fostering human willingness to cooperate [32]. Interestingly, the robot's face does not need to be strictly anthropomorphic. The mere presence of facial expressions can be sufficient to evoke emotional responses from humans, similar to those experienced during interactions with other people [32]. This emotional engagement helps establish a sense of connection between humans and robots, promoting social acceptance in human-populated environments. For example, the work in [33] explores how a robot can signal its motion intent to pedestrians by rotating its head in the direction of movement—both the head and body are controlled

using an artificial potential field (APF). This illustrates how head and face design in robots involves balancing functional requirements (e.g., perception and navigation) with social considerations (e.g., human perception and interaction).

3.2. Pan-Tilt Mechanism

The first design consideration pertains to the pan-tilt mechanism. For this, both its mechanical structure and motors need to be determined. Motor selections are mainly based on torque and acceleration requirements. The required torque is determined considering head weight, including those of its components, motors, and cabling. In order to ensure the safety and longevity of the structure, a safety factor is introduced to the design. This would cover the cable weights and, more crucially, mitigate the structural deprecations that can be introduced from the continuous usage of the PTU. Thus, we would maintain our positioning system's accuracy despite the loads (sensors and screen) on it.

The motors are selected to be Dynamixel DC Servo motors due to their compact design, precise control, and high torque capabilities. The mechanical structure is such that the axes of rotation of the PTU are such that we will not have any offset. The weights of the motors are taken to be 0.25 kg each. The pan-tilt mechanism, including electronics, is assumed to be 0.5 kg. Thus, the resulting overall head weight is approximately 3.2 kg. Using SOLIDWORKS software mass properties, the inertia moments taken on the pan and tilt axis of rotation are determined to be $I_{pan} = 0.019kg.m^2$ and $I_{tilt} = 0.157kg.m^2$, respectively. For acceleration, maximum pan acceleration is taken $\frac{\pi}{2}\ rads/sec^2$ [34]. Since I_{tilt} is much bigger, the required torque $T_{req,H} \approx 0.25N.m$. Twice this value is assumed to be exertible as a moment. A safety factor of 2.5 is then taken as the head mechanism will need to embed expensive sensors like LIDAR safely. It would yield a $1.25N.m$ torque requirement, especially for tilt. On the robot's website, the motor's torque specifications are given with respect to the stall torque (not the rated torque). The respective selection guide states the following: ' *Stall torque is the maximum instantaneous and static torque. Stable motions are possible*

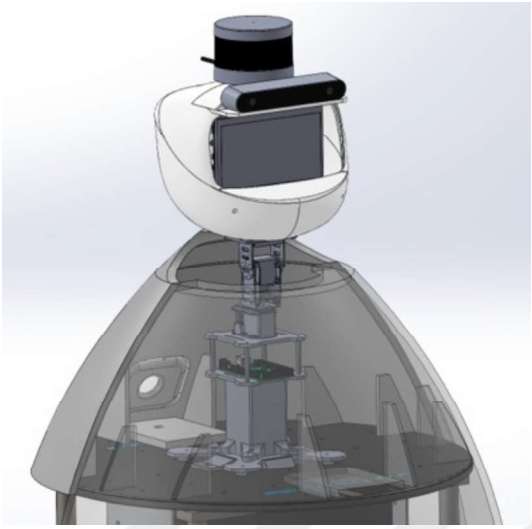
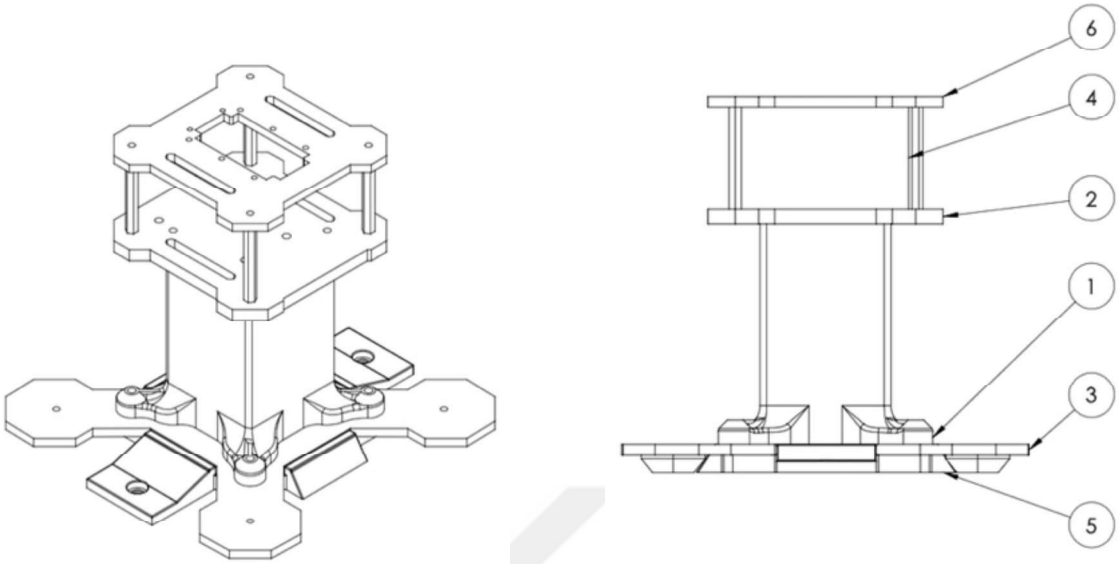


Figure 3.2. SempRob head design.

with robots designed for loads with 1/5 or less of the stall torque.'. Therefore, the motors are required to have around 6.25 N.m stall torque. It is important to note that this represents an overestimation of the required torque, with all safety factors taken into account. Accordingly, we have selected the Dynamixel MX-64T motors, as shown in Figure 3.4(a), based on their technical specifications, which include a stall torque limit of 6.0 N·m at 12 V. Specifications for this motor are given at Table 3.2. The finalized design of the whole head mechanism attached to the robot base can be seen in Figure 3.2. The mechanical structure of the pan-tilt mechanism holding the head is shown in isometric and side views in Figure 3.3(a) and Figure 3.3(b), respectively. The main attachment to the body is done by part 5, shown in the Figure 3.3(b). Similarly, between numbered parts 2 and 6, we are to place the electronics separated with spacers numbered 4. Starting from 6, we mounted the motors and connected the head through brackets.

3.3. Head Electronic Design

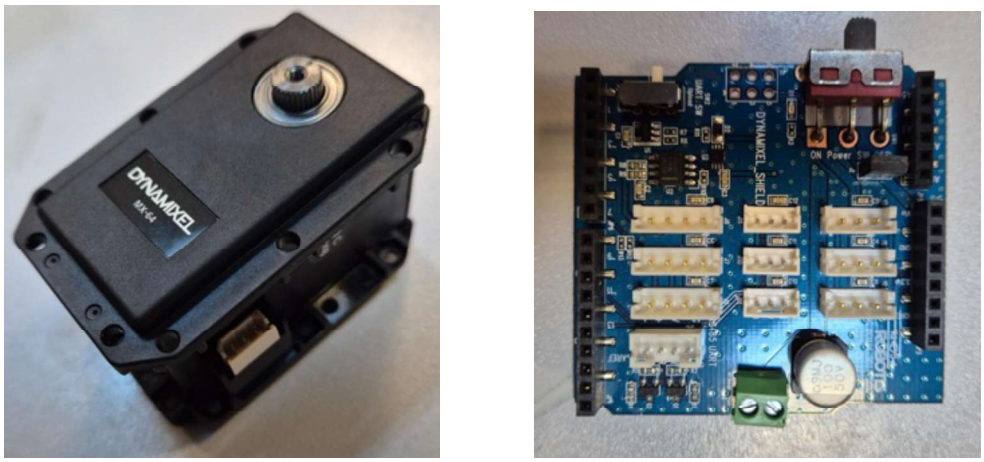
The electronic design aims to accomplish the control of the PTU. A dedicated microcontroller is used for this. As this microcontroller will communicate with the robot's main processor as necessary, all low-level processing associated with motor control is



(a) (b)

Figure 3.3. Head attachment fixture, (a) isometric view (b) side-view.

isolated from the main processor. The selected motors, Dynamixel MX-64T (can be seen in Figure 3.4(a)), communicate via a TTL interface; normally, any TTL-UART interface would work. Specifications for this motor can be seen in Table 3.2. These motors are observed to be capable of a theoretical limit of 6.0 N.m stall torque at 12 V. However, since we are interested in controlling the motor using a microcontroller, we are required to construct the TTL signals as well as the motor driver for voltage regulation and power management. A ROBOTIS product Dynamixel Shield, enables this type of control. The card is shown in Figure 3.4(b). It can draw up to 10 A from its terminal, which is more than double our requirements. This shield supports the Dynamixel Libraries to program the microcontroller and is specifically designed for Arduino products, which is great for rapidly realizing the software structure since it is easy to program and compatible with the Robot Operating System (ROS) framework. There is one caveat: Dynamixel Shield blocks the default UART port of Arduino while driving the motors. Consequently, another serial port from the Arduino needed to be opened for ROS to communicate with the microcontroller. The available serial must be identified specifically for the microcontroller that is being used. For establishing



(a) ROBOTIS Dynamixel MX-64T (b) ROBOTIS Dynamixel Shield

Figure 3.4. (a) Pan tilt mechanism motors and (b) motor driver shield.

communication microcontroller code should specify the serial channel used. We use the ROS serial communication interface in *rosserial_arduino* headers. So, i.e., using Serial1, must specify iostream as Serial1. Each microcontroller board has different Serial port allocations on the PINs. The receive and transmit pinouts of the corresponding serial port must be connected to the UART to USB converter and connected to the Jetson platform. Arduino Mega Rev3 microcontroller is selected because it has 4 separate serial pins that we can use using any USB-UART converter. For this purpose, we have selected to use Sparkfun Serial Breakout CH340C, which can be seen in Figure 3.5(a). Thus, it becomes possible to establish a communication channel to give commands and read the encoder information.

The electronic design is finalized by connecting the head electronics to those of the robot's body. The base platform offers 12 V power output at its Connector I pins. To ensure voltage stability, a DC-DC buck converter is utilized. In particular, the XL4016 card is selected. The card has an input range of 8-40 V and an output range of 1.2-35 V, with a maximum current of 8 A. The selected DC-DC buck converter can be seen in Figure 3.5(b).

Table 3.2. Specifications of ROBOTIS Dynamixel MX-64T.

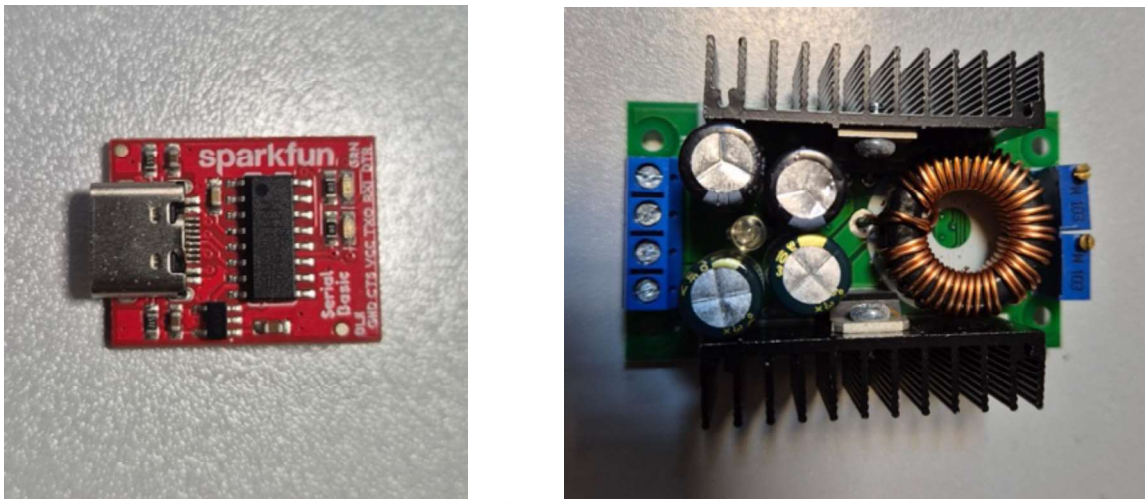
Feature	**Specification**
Stall Torque (N.m)	6.0
No-Load Speed (RPM)	63
Input Voltage (V)	12
Control Algorithm	PID
Resolution (pulse/rev)	4096
Weight (g)	126
Dimensions (m^3)	40.2 × 61.1 × 41
Material	Plastic case, full metal gears
Communication	TTL (3-pin)
Operating Modes	Wheel, Joint, Multi-turn

The overall electrical connections of the SempRob can be seen in Figure 3.6, including the head design components. The components are listed in the Table 3.3, the number column refers to numbered components in the Figure 3.6.

3.4. Head Control

The goal of head control is to align the optical center of the incoming RGB image with the object of - namely the human target. Let $f \in \mathcal{F}$ defined as $f = \begin{bmatrix} f_1 & f_2 \end{bmatrix}$ denote the pan and tilt angles of the pan-tilt mechanism of the head. Here, the set $\mathcal{F} \subset S^2$ denotes the configuration space of this mechanism. It is defined as $\mathcal{F} = [-\frac{\Delta_1}{2}, \frac{\Delta_1}{2}] \times [-\frac{\Delta_2}{2}, \frac{\Delta_2}{2}]$ with horizontal FOV $\Delta_1 = \frac{2\pi}{3}$ and vertical FOV $\Delta_2 = \frac{4\pi}{9}$.

Consider $N_1 \times N_2$ size RGB-D frames. Let $x^h = \begin{bmatrix} x_1^h & x_2^h \end{bmatrix}^T$ denote the human position in the RGB image and $d(x^h)$ be the respective depth reading. Let $f_1(x^h) \in [-\frac{\Delta_1^C}{2}, \frac{\Delta_1^C}{2}]$ denote the pan offset of target x^h and $f_2(x^h) \in [-\frac{\Delta_2^C}{2}, \frac{\Delta_2^C}{2}]$ denote the vertical offset of the target from optical center where the camera has horizontal FOV Δ_1^C and vertical FOV Δ_2^C. Note that the former angle encodes the goal pan direction while



(a) (b)

Figure 3.5. (a) Sparkfun serial breakout CH340C, and (b) XL4016 DC-DC buck converter.

the latter encodes the goal vertical direction for the pan-tilt mechanism so that the camera's optic center becomes aligned with the object of interest. The angle $f_1(x^h)$ can be computed as

$$f_1(x^h) = \left(\frac{x_1^h - \frac{N_1}{2}}{\frac{N_1}{2}} \right) \left(\frac{\Delta_1^C}{2} \right). \tag{3.1}$$

Note that the planar estimate of x^h in odometry frame is denoted as $\hat{x}^h_{odom}$ and is obtained by

$$\hat{x}^h_{odom} = d(x^h) \begin{bmatrix} \cos\big(f_1(x^h)\big) \\ -\sin\big(f_1(x^h)\big) \end{bmatrix}. \tag{3.2}$$

Similarly, the tilt goal angle $f_2(x^h)$ can be computed. For the head to look at goal direction $f(x^h) = \begin{bmatrix} f_1(x^h) & f_2(x^h) \end{bmatrix}^T$, a velocity-based control approach is proposed - namely

$$\dot{f} = \omega_H. \tag{3.3}$$

Table 3.3. SempRob electrical diagram component list.

Number	**Component name**
1	StereoLabs ZED 2 Stereo Camera
2	Nvidia Jetson Xavier AGX
3	XL4015 5A Adjustable DC/DC Voltage Stepper
4	Waveshare 7" Capacitive Touch Screen LCD (H)
5	Robosense RS-Lidar 16
6	Segway RMP 220
7	Dynamixel MX-64T
8	XL4016 DC-DC Max 9A Step Down Buck Converter
9	Dynamixel Shield
10	Sparkfun serial breakout CH340C
11	Arduino Mega 2560

The velocity input ω_H is obtained through defining an artificial potential function $\varphi_h : \mathcal{F} \to R$ as follows:

$$\varphi_h(f) = \begin{bmatrix} (f_1 - f_1(x^h))^2 \\ (f_2 - f_2(x^h))^2 \end{bmatrix}^{k_H} . \tag{3.4}$$

In Equation 3.4, the parameter $k_H \in R$ is an urgency term. The gradient of φ_h with respect to f is given by

$$D_f\varphi_h(f) = -2k_H \begin{bmatrix} (f_1 - f_1(x^h)) \\ (f_2 - f_2(x^h)) \end{bmatrix}^{(k_H-1)} . \tag{3.5}$$

Finally, the velocity commands for the PTU are computed with using Equation 3.5 as

$$\omega_H = \begin{bmatrix} \omega_{1,max} & 0 \\ 0 & \omega_{2,max} \end{bmatrix} D_f\varphi_h(f). \tag{3.6}$$

In Equation 3.6 $\omega_{1,max}, \omega_{2,max}$ terms are the maximum angular velocities of the pan and tilt motors, respectively. In equations, we used the notation H, for head and h, for where we used the target human pose.

3.5. Head Control Software

The flow of processing for the pan and tilt control is given in Figure 3.7. Environmental sensing is based on the RGB-D data. For human detection, a YOLOv4 Human-Gesture network that was trained previously within our laboratory was used [35]. Depth readings within the bounding box of the target human are used to estimate the human's planar position relative to a fixed odometry frame. This estimate is continuously transformed to compensate for the robot's translational motion and pan-tilt head rotations. A Kalman filter is used for estimating the states of the human target, namely, planar position and velocities (4 states total). This estimation is for recovering the human target in case the target is lost.

The low-level structure is summarized in pseudo-code outlined in Algorithm 1. It is specifically designed to modify designated motor memory addresses during operation, enabling dynamic changes to the motor's driving modes for full behavioral control as needed. Additionally, it supports simultaneous control of both motors and outputs encoder readings via a custom message for high-level processing, thereby closing the feedback loop.

3.6. Robot Face

The robot face is simulated using an open-source robot face code [36]. This code allows one to alter expressions and gaze at a given point. Those capabilities are slightly changed to improve the responsiveness of a given target. The code was adapted for its appearance to be socially plausible as seen in Figure 3.8. Following, it was integrated with the human detection result as to have gaze locations and expressions based on interaction with the human.

3.7. Head and Robot Body Integration

The realized robot head assembly can be seen in Figure 3.9. For head body integration, a head bed is designed as seen in Figure 3.10(a). This bed is placed on top of the robot's top plate. It is attached to this place using the grid of holes. The final form of SempRob can be seen in Figure 3.10(b).

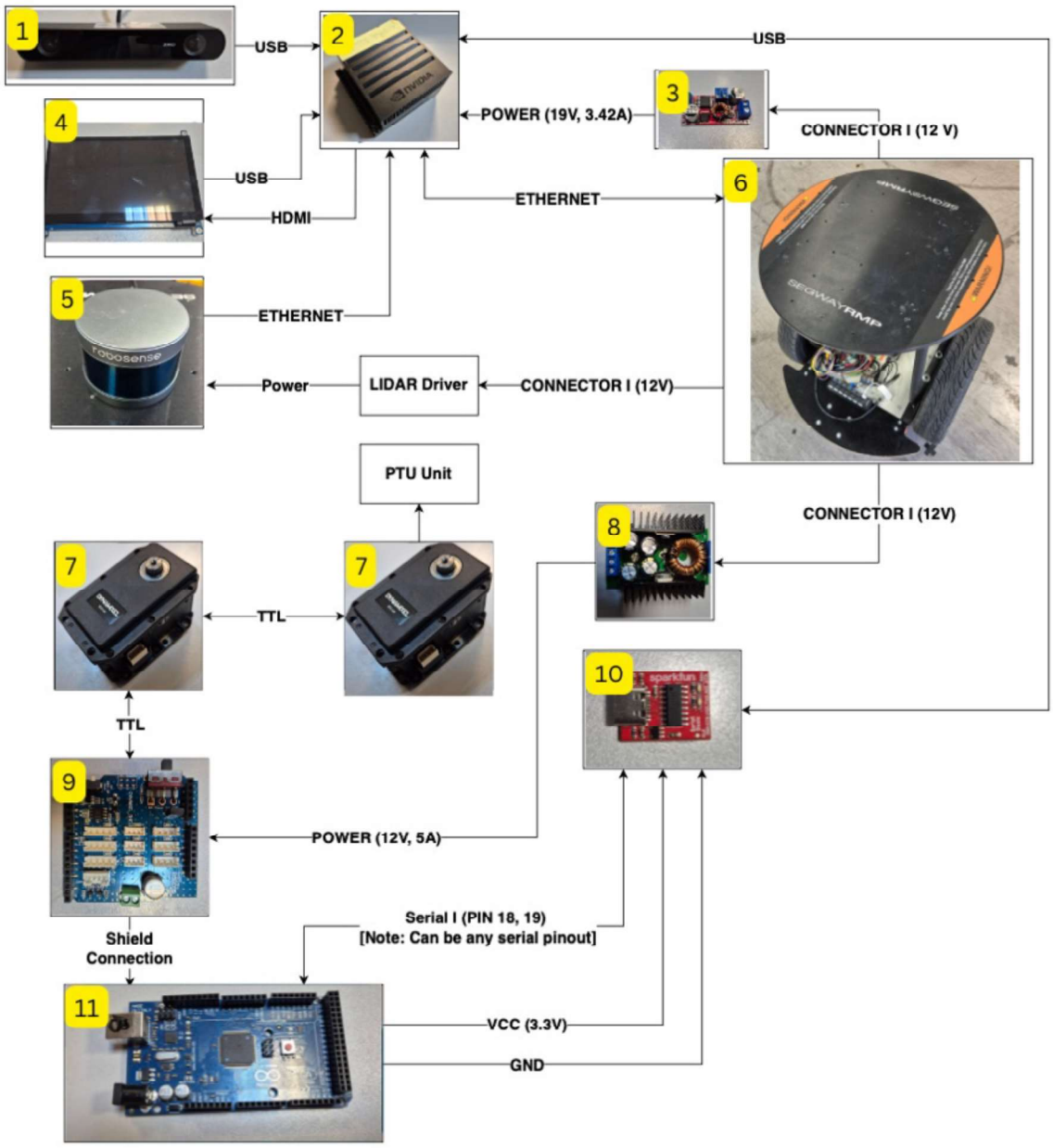


Figure 3.6. Electrical diagram and communications of the SempRob.

Algorithm 1 Low-Level PTU Control Flow

Initialize:

Configure motor constants (IDs, baud rate, control mode)

Set up hardware (serial communication, GPIO, etc.)

procedure SETUP

Configure motor driver (baud rate, operating mode)

Establish ROS communication

Enable motor torque

end procedure

procedure CONTROLCALLBACK(*msg*)

if Position control mode **then**

Set motor positions based on *msg*

else if Velocity control mode **then**

Set motor velocities based on *msg*

end if

end procedure

procedure FEEDBACKLOOP

while System is active **do**

Read motor states (position, velocity)

Send feedback to ROS node

Check for errors or communication issues

Delay for loop stability

end while

end procedure

procedure ERRORHANDLING

if Error detected **then**

Disable motor torque

Report error to ROS node

end if

end procedure

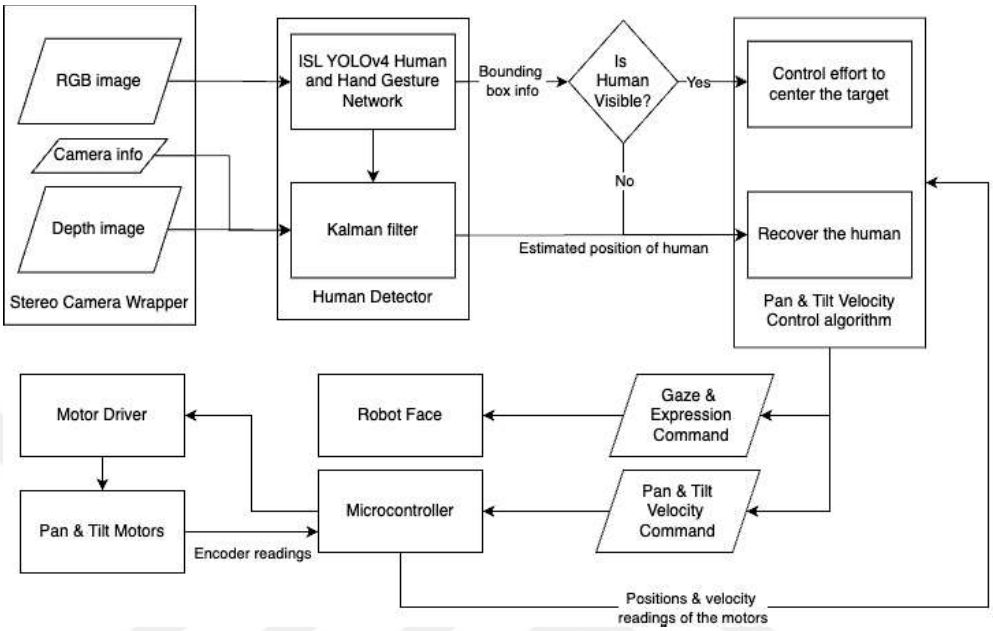


Figure 3.7. Flow of processing for head control.

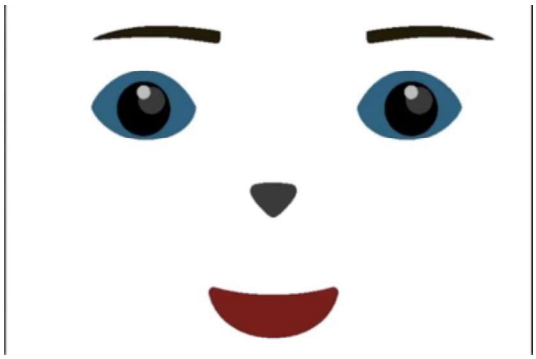


Figure 3.8. Robot face simulation [36].

Figure 3.9. SempRob head and neck assembled.

(a) (b)

Figure 3.10. SempRob, (a) head attachment and head rest, (b) final assembly with head attached.

4. HUMAN FOLLOWING AND BODY-HEAD COORDINATION

This chapter addresses the problem of human-following, a fundamental capability in human-robot interaction, particularly within social robotics applications [25]. A common baseline is the decoupled approach, where the robot navigates using Social APF-RL while its head independently attempts to maintain visual focus on the human target. This approach is termed "decoupled" because the reinforcement learning (RL) policy does not incorporate head movement into its decision-making process. In contrast, a coupled approach introduces coordinated control of both the robot's body and head during navigation. By integrating body-head coordination into the RL framework, the robot can learn socially compliant human following behaviors that exploit the head mechanism for improved perception, thereby ensuring relatively less interrupted and thus more robust target tracking. The proposed method builds upon previous work in social navigation and human-following [4], with the key distinction being the integration of body-head coordination into the socially compliant navigation model.The contributions of this chapter can be summarized as follows:

- Social APF-RL navigation has been used for human following by a mobile robot.
- The obstacle generator was updated — sourced from another ongoing project— to better model environmental obstacles.
- Social APF-RL for Human Following is proposed for achieving socially compliant and safe navigation via simultaneous head movement - albeit in a decoupled manner.
- Head-Coordinated Social APF-RL for Human-Following is proposed as a novel method for achieving socially compliant and safe navigation, while also minimizing the likelihood of losing the target via coordinating body-head movements.

The outline of the chapter is as follows: Related literature is discussed in Section 4.1. Following, we present revisions in the formulation of the original Social APF-RL

to improve performance with the decoupled approach in Section 4.2. Then we will discuss the usage of our revised navigation method and head mechanism simultaneously for human following task in Section 4.2.2. Following, the learning of the body-head coordination is explained in Section 4.3. This section consists of examining the environmental complexity in Subsection 4.4, and by that complexity analysis, the developed and used training environment is presented in Subsection 4.6. Lastly, the test results of the coordination model we trained in Section 4.7.

4.1. Related Work

First, we present a brief review of deep learning based human following. It should be noted that in most work, learning is typically done in a simulation environment. In these work, the Social Force Model (SFM) is used for generating realistic human trajectories [16]. For completeness, a brief explanation of SFM is provided in Appendix C. Next, we review work on human following considering pan-tilt head mechanisms.

4.1.1. Deep Learning Based Human Following

Several reinforcement learning-based approaches have been proposed to address the human-following problem in social robotics. One such method is a Markov-based approach introduced in [37], where the robot infers the goal of the target human rather than simply following their trajectory. This enables the robot to better handle unpredictable human motion. The method is integrated with an on-policy reinforcement learning algorithm, SARSA, to learn a human-following policy. The learning process employs tabular Q-values and an ε-greedy exploration strategy, with ε set to 0.4 during training and reduced to 0.05 during evaluation. The reward function encourages the robot to maintain a close distance to the human while minimizing the overall path length. The action space is discretized into eight directions, and training includes three different goal locations with 20 diverse human trajectories.

A hybrid deep reinforcement learning (DRL) method is presented in [38], combining supervised learning with DRL to incorporate prior task knowledge. The approach starts with a CNN-based supervised policy network trained on a combination of real and synthetic images. These images are labeled with discrete actions such as turn left, go forward, and turn right. The policy learned from supervised learning is then transferred to DRL models—specifically, Deep Q-Network (DQN) and Advantage Actor-Critic (A2C). To facilitate this transfer, the DRL architectures are kept consistent with the CNN used in supervised learning, except for the removal of the softmax layer in the value networks. The system infers the target human's velocity by stacking four consecutive frames as input. Stochastic gradient ascent is used for supervised training, while the Adam optimizer with a learning rate of 10^{-6} is applied for DRL policy learning. This hybrid approach is designed to initialize DRL with heuristic knowledge from supervised learning, thereby improving learning efficiency and performance.

Another end-to-end DRL approach is proposed in [39], utilizing the Socially Attentive Reinforcement Learning (SARL) architecture [40]. The reward function incorporates success incentives, collision penalties, goal-directed rewards, and obstacle avoidance. Observations include 2D laser scan data and the relative position of the goal. To enable social navigation, additional social state information is introduced for seven surrounding pedestrians. Training is conducted in a simulated environment using Gazebo and the Arena-Rosnav framework [41,42]. Human trajectories are modeled using an adapted version of the Social Force Model (SFM) with the Pedsim plugin. During evaluation, the agent and dynamic obstacles have maximum velocities of 0.22 m/s and 0.3 m/s, respectively. Although these speeds are lower than average human walking speed, they are justified by the high density of the environment, which includes 20 dynamic obstacles.

4.1.2. Human Following with Pan-Tilt Head Mechanisms

Two distinct algorithms for mobile robots equipped with head mechanisms are proposed in [43]. The first leverages Kinect's skeleton tracker to replicate a human's

path in real time, while the second employs global planning in environments with pre-existing maps. Notably, their primary tracking strategy is entirely dependent on the Kinect skeleton tracker, making both approaches viable only when the robot's camera is positioned higher than surrounding obstacles. Consequently, these methods lack a mechanism for recovering the human target in cases of occlusion.

In [44], human-following is achieved using a monocular camera mounted on a mobile robot with a pan-tilt head. The system employs the tinyYOLO detector to identify humans, and subsequently uses Flow-track++, a Siamese network-based tracker, for continuous tracking. To handle occlusions or cases where the target moves out of view, an iterative local-to-global search strategy based on the Siamese network output is used for target recovery. A unified controller directs the pan-tilt mechanism to keep the camera aligned with the center of the human's bounding box horizontally, while maintaining a predefined vertical offset. A different approach, presented in [28], frames the human-following task as a pursuit-evasion game, where the robot—also equipped with a pan-tilt head—actively tracks a dynamic target.

Interestingly, few studies explore navigation with coordinated body-head control. A notable exception is found in [45]. In this work, the OptiTrack motion capture system is used for human localization, while a directional-cost-based local planner ensures smooth navigation and obstacle avoidance. The robot solves a multi-criteria decision-making problem using the Analytical Hierarchy Process (AHP) to determine how best to utilize its behavioral modules during navigation. Two specific head behaviors are defined. In look-at-path, the robot turns its head toward local planner goals, signaling its intended path to nearby humans. In glance-at-human, the robot ranks surrounding humans by their positions and velocities, then sequentially turns its head toward each person without repetition. This behavior is designed to communicate awareness and acknowledgement, contributing to human comfort and social acceptance. Importantly, since the robot's navigation goal is static and known via a global map, the robot can afford to divert its gaze without losing track of its main objective.

While this work shares similarities with the approach proposed in this chapter, it differs in two critical ways. First, it does not address human-following, which is central to the proposed method. Second, it relies on a global planner for obstacle avoidance, in contrast to the reactive obstacle avoidance strategy used in our approach.

4.2. Social APF-RL for Human Following

Social APF-RL for Human Following builds upon the Social APF-RL navigation approach. It should be noted that Social APF-RL navigation can easily be used for human following by considering a dynamic goal. Rather than considering a stationary target location, the target is now moving. Furthermore, since the target is a human, the robot maintains its distance to its target as it is navigating towards its goal location.

However, when human-following is required across large or multi-room environments, Social APF-RL navigation becomes becomes inadequate. This is because the definition of the APF assumes that the workspace is confined with known or detectable outer limits. With this assumption, the outer boundary of the room was modeled as a single large ellipse derived from the convex hull of the respective laser data. Interior obstacles were identified by clustering the remaining data points and fitting ellipses to each resulting cluster. However, this approach proves inadequate for continuous navigation in multi-room environments. Modeling the outer boundary with a single ellipse prevents the representation of transitional architectural elements such as doorways or hallways between rooms. As a result, the assumption of a fixed enclosing boundary is no longer valid, necessitating a more flexible and adaptive obstacle modeling approach.

4.2.1. Social APF-RL for Human Following: Theoretical Revisions

For this, the original Social APF-RL navigation is revised as to enable human following. We refer to this approach as ‘Social APF-RL for Human Following’. This approach considers an updated obstacle generator — sourced from another ongoing project — as to model obstacles encountered as the robot is navigating in a manner

suitable for human following while not making such an assumption. For completeness, the new obstacle generation method is explained in Appendix A. Accordingly, the definition of the artificial potential function φ had to be revised. In particular, the following modifications were introduced:

- Due to the new obstacle model, the denominator term no longer encodes the outer boundary as an obstacle.
- The encoding of each obstacle is associated with a weight as defined by its exponent. The motivation for this is that since an outer boundary cannot be assumed, the number of obstacles increases considerably. As the artificial potential function encodes each obstacle, the resulting vector fields need to be more carefully maneuvered. This may be alleviated by prioritizing the obstacles based on their distances as defined the exponent terms. This is explained in Appendix B.1.
- Finally, in the previous version of APF-RL and Social APF-RL, the definition of APF function becomes undefined if the denominator term becomes zero. This is addressed via making the artificial potential function admissible [46] as detailed in Appendix B.2.

4.2.2. Social APF-RL for Human Following: Experimental Evaluation

The proposed Social APF-RL for Human Following approach is tested in simulation in the task of following a human along a corridor case as shown in Figure 4.1. The human actor follows a zigzag path along the corridor. The human goal is given on a fixed odometry frame using the state estimate obtained through the same Kalman Filter as discussed in Section 3.4. Performance is compared with the original version of Social APF-RL based navigation as shown in Figure 4.2; in the Figure 4.2(a) gives the human and robot path in the old versions, while the Figure 4.2(b) gives out the revised one. While turquoise paths correspond to those of the human actor, the robot's paths are indicated by the green-red paths. In the green parts of this path, the target human is visible to the robot, while in the red parts, it has lost the human target and is trying to recover the visual of the human. The scan points can be seen as magenta dots, and

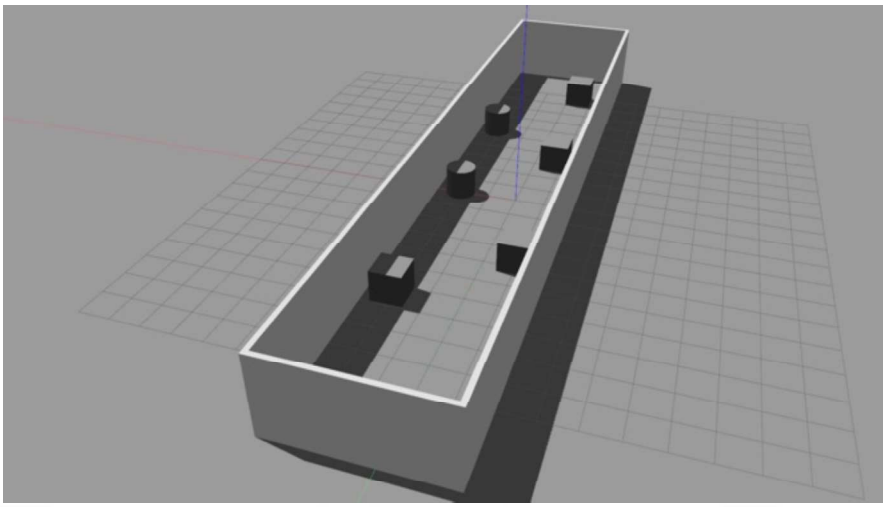
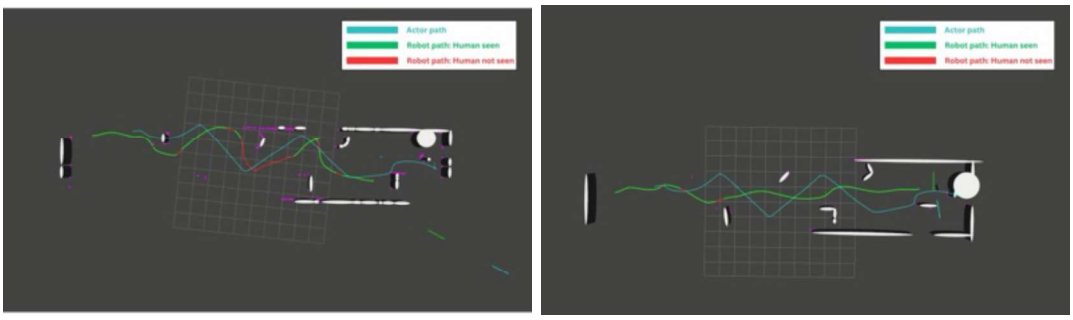


Figure 4.1. Simulation corridor environment.

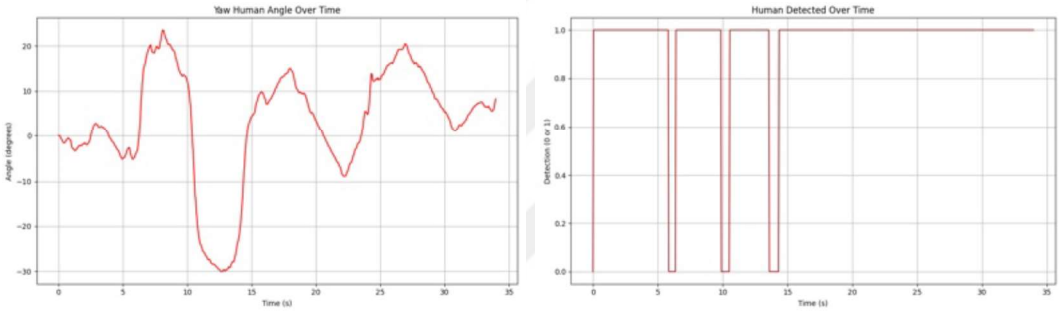
on top of them are elliptically fitted obstacles shown as elliptic cylinders with white color. We can observe that with Social APF-RL navigation with a moving target, the modeling of the corridor walls with a single ellipse causes the robot to move sideways at high angular velocities, causing it to lose sight of the human target, creating inconsistencies. On the contrary, the proposed Social APF-RL Human Following approach results in a much more stable following path with minimal loss of sight of the target. Upon examining the yaw angle of the human with respect to the mobile robot heading as shown in Figure 4.3(a), this angle is observed to be between $[-30°, 30°]$. In parallel, robot's loss of human and recovery is as seen in Figure 4.3(b). As expected, the high yaw angle changes align with the robot loses and then recovers the human target. Since the robot's body and head mechanisms are moving independently of each other, this is mainly due to the body's obstacle avoidance, resulting in unexpected rotations while the head tries to keep the human target at its center.

In summary, Social APF-RL Human Following enables reduced loss of sight of the target. However, as the body and head are moving independently of each other, minimizing the yaw angle of the human with respect to our robot cannot be ensured. While this can be partially achieved by increasing the head mechanism speed, higher speeds - especially the pan speed - sometimes results in oscillatory motion due to



(a) (b)

Figure 4.2. Corridor human following robot and human paths, using (a) old obstacle generator, (b) revised obstacle generator.



(a) Robot's alignment to the target human (b) Human target detection

Figure 4.3. Performance of Social APF-RL for human following in a corridor.

the frequency of the camera input and the algorithm rate. Also, the robot's body can make jerky movements while traversing through cluttered and human-populated environments as to avoid collisions with them.

4.3. Head-Coordinated Social APF-RL for Human Following

The objective is to enable the robot to follow a human while maintaining visual contact as consistently as possible, and simultaneously adhering to social navigation norms. We propose a method that couples head movements with body movements, thereby establishing a body-head coordination model. This coordination offers several key benefits:

(i) It enables human-like navigation behavior, where the robot first orients its head toward the target before initiating movement, mimicking natural gaze-led motion,

(ii) It helps maintain continuous visual contact with the target (except in cases of occlusion or obstruction), resulting in more precise and reliable human-following,

(iii) It improves obstacle avoidance by aligning head and body orientations, thereby enabling smoother and more efficient motion through coordinated angular velocities.

The proposed approach integrates a deep learning framework to model body-head coordination. Specifically, the learning architecture used in the Social APF-RL method (detailed in Appendix B) for human-following tasks is extended to incorporate head movement dynamics. The learning process is formulated as a Partially Observable Markov Decision Process (POMDP) and employs Deep Reinforcement Learning (DRL). We adopt a Soft Actor-Critic (SAC) network architecture consisting of four fully connected layers, each with 512 neurons. The overall pipeline for training and deploying the body-head coordination model is illustrated in Figure 4.4. The model has been adapted to learn coordinated body and head motion with a moving human target. In particular, the following are changed:

- Action set
- Input observations
- Reward functions

4.3.1. Action Set

First, the action set is revised as:

$$a''(t) = [k(t), g^T(t), \mu_h(t), k_H(t)]. \tag{4.1}$$

The ranges of the action vector's members given in Equation 4.1 are given in Table 4.1. The $k(t)$ is the learned function for the power term that regulates the attractive

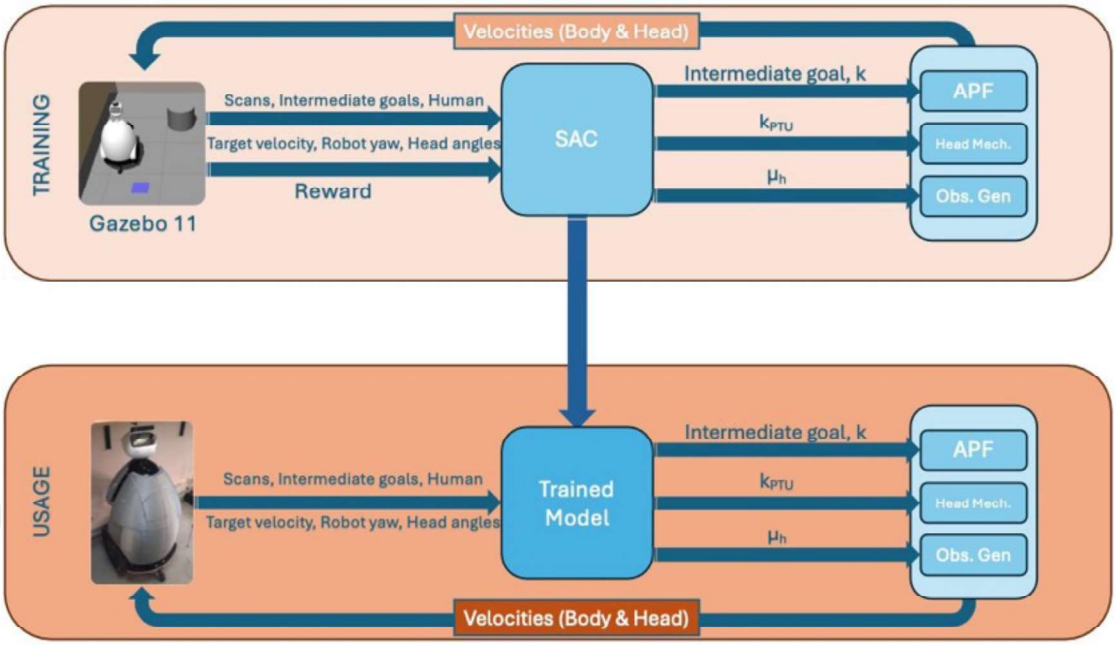


Figure 4.4. Body-head coordination model training and deployment pipeline.

field of the APF navigation function we use along with our DRL model. The $g^T(t) = [g_1(t), g_2(t)]$ is the function we learned to give the goal target to the APF navigation module based on the current observation space. The $\mu_h(t)$ is a learned function for providing the scalar that determines how the enlargement of the elliptic human obstacle representations will be carried out. Lastly, the $k_H(t)$ is similar to the first action, but it provides the power term for the head mechanism APF module to control the speed of the pan and tilt motors.

4.3.2. Observation Space

Subsequently, the observation space has been revised. The revised observations $\omega(t)''$ consist of the following components:

$$\begin{aligned}\omega(t)'' = (&q(t), q(t-\delta t), g_h(t), g_h(t-\delta t), q_h'(t),\\ &v_{h_{\text{target}}}(t), v_{h_{\text{target}}}(t-\delta t), q_{\text{camera}}(t), \alpha_r(t), \alpha_r(t-\delta t), f(t), f(t-\delta t)).\end{aligned} \tag{4.2}$$

Table 4.1. Body-head coordination actions and their ranges.

Action	**Range**
$k(t)$	$[2.0,\ 10.0]$
$g_1(t)$	$[0.0,\ 1.25]$
$g_2(t)$	$[-0.75,\ 0.75]$
$\mu_h(t)$	$[0.0,\ 1.0]$
$k_{\mathrm{H}}(t)$	$[1.0,\ 5.0]$

The first two components encode the current laser scan $q(t)$ along with the previous scan $q(t-\delta t)$. Both have angles quantized to N levels and thus each reading $q(t)[i] \geq 0$ represents the depth measurement from angle $(i - \frac{N}{2})\delta f_1^L$ where δf_1^L is the laser's pan resolution. For identifying the target, we used a modular approach where the estimated pose of the human target is input to the network. The following two inputs encode detected human data including the human target.

One approach to enable the network to distinguish and follow the target human would be to input an image annotated with bounding boxes around detected humans. However, this significantly increases the learning complexity and training time. As a more efficient alternative, we modify the binary human label scan used in the Social APF-RL navigation framework. The modified scan introduces a tertiary labeling scheme to differentiate between generic obstacles, non-target humans, and the specific target human that the head mechanism should follow. Specifically, the scan labels are redefined as follows:

0 indicates a generic obstacle,
1 indicates a human (not the target),
2 indicates the target human

The resulting human label scan is denoted as:

$$q'_h(t) \;=\; (q'_h(t)[1],\, \ldots,\, q'_h(t)[N])\,, \tag{4.3}$$

where, $q'_h(t)[k] \in \{0, 1, 2\}$ and N denotes the scan size of the laser scan. The human target's estimated planar velocity is also fed as an observation ($v_{h_{\text{target}}}(t)$) in order to ensure better following. The human target's previous estimation is also added ($v_{h_{\text{target}}}(t - \delta t)$) in order to infer the change in direction of the movement.

In order to encode whether the target is visually visible or not, the network is also input a scan that designates portions of the laser data that overlap with the camera's field of view. This observation is defined as follows:

$$q_{\text{camera}}(t) = \begin{cases} 1 & \text{If } (i - \frac{N}{2})\delta f_1^L \in \mathcal{F}_1 \text{ and } q(t)[I] < D_{max} \\ 0 & \text{Else} \end{cases}, \tag{4.4}$$

where, $\mathcal{F}_1 = [-\frac{\Delta_1^C}{2}, \frac{\Delta_1^C}{2}]$ and $D_{max} = 15$ based on lidar's depth sensing range.

Robot's current and previous yaw angle ($\alpha_r(t)$ and $\alpha_r(t - \delta t)$) estimated with respect to a fixed odometry frame are also input so that the network is informed of the body orientation.

The current and previous pan and tilt motor encoder readings ($f(t) = [f_1(t), f_2(t)]^T$ and $f(t - \delta t) = [f_1(t - \delta t), f_2(t - \delta t)]^T$) are also input so that the model is informed of the head orientation. This is for the model to interconnect the observations of those angles with the aforementioned scans to learn the action function, $k_H(t)$, that will quantitatively moderate the attractiveness of the target human for the head mechanism.

All observation vectors except the tertiary label of humans are normalized. The scan points are normalized with respect to their maximum measurement range. So

overall, the network given the downsampled scan size to be equal to 32 is: 142 x 512 x 512 x 512 x 512 x 512 x 5.

4.3.3. Reward Function

The reward function is redesigned to facilitate the learning of actions, enabling the network to effectively develop the desired coordination behavior. Recall that the robot's planar position is defined by $c(t) \in R^2$ while its heading is defined by $\alpha(t) \in [-\pi, \pi]$. Consider a human target at location $g^h(t) \in R^2$. Let u_{hr} encode the direction of the target human with respect to our robot (egocentric):

$$u_{hr}(t) = c(t) - g^h(t). \tag{4.5}$$

Furthermore, let $\chi_{dh}(t)$ denote whether the target human is detected or not. Its definition is given as

$$\chi_{dh}(t) = \begin{cases} 1 & \text{if target detection} \\ 0 & \text{Otherwise} \end{cases}. \tag{4.6}$$

Let $\theta_h \in [-\pi, \pi]$ denote human target's angle with respect to the robot:

$$\theta_h = tan^{-1}\left(\frac{u_{rh_2}}{u_{rh_1}}\right), \tag{4.7}$$

where $u_{rh} = -u_{hr}$, given in Equation 4.5. Robot's heading error $e_{r\alpha}$ and pan error $e_{f_1\alpha}$ with respect to target human can be calculated as follows:

$$e_{r\alpha}(t) = \left(\theta_h(t) - \alpha_r(t)\right), \text{ and} \tag{4.8}$$

$$e_{f_1\alpha}(t) = \left(\theta_h(t) - (\alpha_r(t) + f_1(t))\right). \tag{4.9}$$

We define a follow window behind the target human in the range of $\tau_{fw}^{+} = 1.7$m to $\tau_{fw}^{-} = 3.6$m with an angular span of $\frac{2\pi}{3}$. Recalling that $v_{h_{\text{target}}}(t)$ refers to the target's

velocity, let $\theta_{fw}(t)$ denote the follow angle:

$$\theta_{fw}(t) = \big((v_{h_{target_2}}(t), v_{h_{target_1}}(t)) + \pi\big) - (u_{hr_2}(t), u_{hr_1}(t)). \tag{4.10}$$

The revised reward function is defined as follows:

$$\begin{aligned} r''(a''(t), \omega''(t)) = {} & r_{fg}(a''(t), \omega''(t)) + r_{fw}(a''(t), \omega''(t)) \\ & + r'_o(a''(t), \omega''(t)) + r_h(a'(t), \omega'(t)) + r_{dh}(a''(t), \omega''(t)) \\ & + \chi_{dh}\big[r_{h\alpha}(a''(t), \omega''(t)) + r_{\varphi_H}(a''(t), \omega''(t)) + r_{f_1}(a''(t), \omega''(t))\big]. \end{aligned} \tag{4.11}$$

It consists of several terms. The goal of these reward terms is to ensure that the robot follows the human while being in the follow window behind the human and having linear movement as much as possible. The intermediate goals are defined in front of the robot with the same goal in mind.

• The first term encodes the goal reward function r_{fg}. It is given as follows:

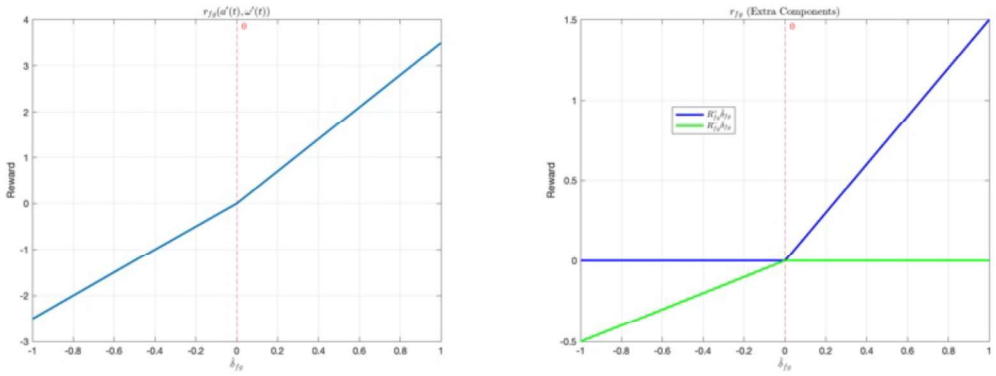
$$r_{fg}(a''(t), \omega''(t)) = \begin{cases} (R_{fg} + R^+_{fg})\dot{\delta}_{fg} & \text{if} \quad \dot{\delta}_{fg}(t) > 0 \\ (R_{fg} + R^-_{fg})\dot{\delta}_{fg} & \text{if} \quad \dot{\delta}_{fg}(t) \leq 0 \end{cases}. \tag{4.12}$$

Here, the term $R_{fg} = 2.0$ is a constant reward term. Its value changes depending on the terms $R^+_{fg} = 1.5$ and R^-_{fg} as well as $\dot{\delta}_{fg}(t)$. The term R^-_{fg} is defined as:

$$R^-_{fg} = \begin{cases} 0.5 & \text{if} \quad \tau^-_{fw} \leq ||u_{rh}(t)|| < \tau^+_{fw} \quad \text{and} \quad \mid \theta_{fw} \mid \leq \frac{\pi}{3} \\ 0 & \text{Otherwise} \end{cases}. \tag{4.13}$$

Recalling that $c \in \mathbb{R}^2$ denotes robot's planar location, the term $\dot{\delta}_{fg}(t)$ is defined as

$$\dot{\delta}_{fg}(t) = \frac{||c(t - \delta t) - g_h(t - \delta t)|| - ||c(t) - g_h(t)||}{dt}. \tag{4.14}$$



(a) (b)

Figure 4.5. Reward function, $r_{fg}(a''(t), \omega''(t))$, (a) base reward, (b) extra reward based on $\dot{\delta}_{fg}$.

The visualization of this reward function can be seen in Figure 4.5. The base reward is shown in Figure 4.5(a), whereas the extra reward (or penalty in case of $\dot{\delta}_{fg}(t) \leq 0$) is given in Figure 4.5(b).

• The second term r_{fw} encodes on whether the robot is within the follow window or not. It is defined as follows:

$$r_{fw}\big(a''(t), \omega''(t)\big) = \begin{cases} 0 & \text{if } \|u_{rh}(t)\| < \tau_{fw}^{-} \text{ or } \theta_{fw} > 60^{\circ} \\ R_{fw}\left(1 - \dfrac{\delta_{fg}(t) - \tau_{fw}^{-}}{\tau_{fw}^{+} - \tau_{fw}^{-}}\right) & \text{if } \tau_{fw}^{-} \leq \|u_{rh}(t)\| < \tau_{fw}^{+} \text{ and } \theta_{fw} \leq \frac{\pi}{3} \end{cases} . \tag{4.15}$$

The visualization of the follow window and and robot that follows the target human can be seen in the Figure 4.7. There are two gimmick models (visualization purpose) present in the simulation, like the follow window, the first one is the blue rectangle indicates the current $g^{T}(t)$, and the red rectangle is the perceived human target goal.

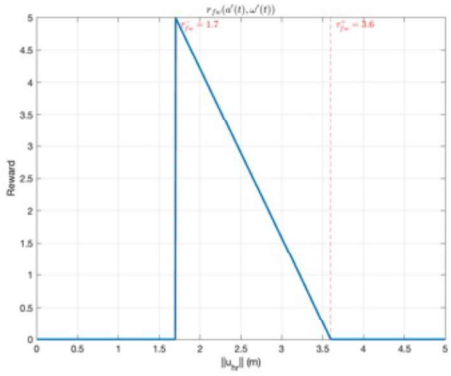


Figure 4.6. Reward function, $r_{fw}(a''(t), \omega''(t))$.

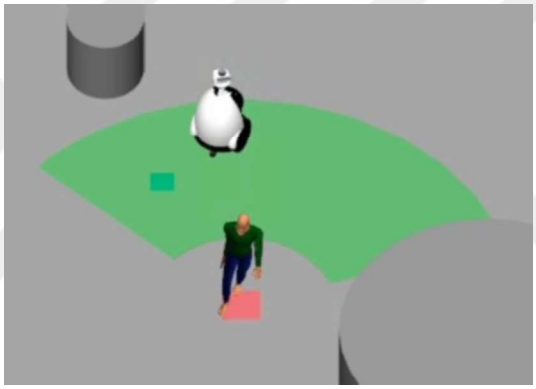


Figure 4.7. Follow window visualization and SempRob following the target.

• The third term r_o considers obstacle avoidance:

$$r'_o(a''(t)\omega''(t)) = \begin{cases} \frac{\beta(c,o)-\tau_o}{R_{o1}} & \text{if } \exists\, o \in \mathcal{O}(t) \text{ such that } 0 < \beta\big(c,o\big) < \tau_o \\ R_{o2} & \text{if } \exists\, o \in \mathcal{O}(t) \text{ such that } \beta\big(c,o\big) = 0 \end{cases}. \tag{4.16}$$

In Equation 4.16, the weight value is set as $R_{o1} = 15$. Hence, if the robot is close to an obstacle as measured by the $\beta(c, o)$ term, it is penalized. The threshold term of obstacle proximity is set to be $\tau_o = 1.7$m. The weight $R_{o2} = -40$ encodes penalty in case of a collision. Thus, as the robot approaches an obstacle, , the penalty increases gradually. On the extreme case of collision, the penalty is increased greatly. The visualization of this penalty function can be seen in Figure 4.8.

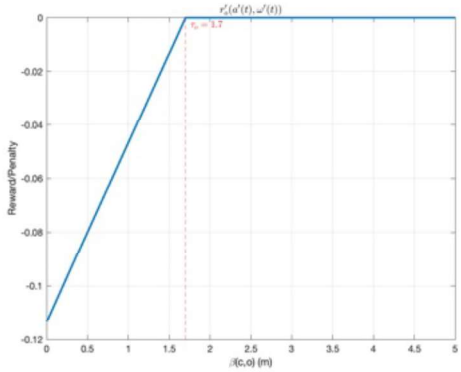


Figure 4.8. Reward function, $r'_o(a''(t), \omega''(t))$.

• The fourth term r_h is associated with social compliance. There is a penalty if a robot comes near any human. It is defined by the following,

$$r_h(a'(t), \omega'(t)) = \sum_{i=1}^{N_h} r_{hi}\big(a'(t), \omega'(t)\big). \tag{4.17}$$

Here, $r_{hi}\big(a'(t)$ encodes the distance of the robot to the ith human as:

$$r_{hi}\big(a''(t), \omega''(t)\big) \;=\; \begin{cases} R_h\left(\tau_h - \left\| c(t) - h_i(t) \right\|\right) & \text{if } \left\|c(t) - h_i(t)\right\| < \tau_h \\ 0, & \text{Otherwise} \end{cases}. \tag{4.18}$$

The parameters in Equation 4.18 are $R_h = 1.5$m and $\tau_h = 1.7$m respectively. This penalty function's visualization can be seen in Figure 4.9.

• The next term r_{dh} penalizes the robot whenever it loses the sight of its target:

$$r_{dh}(a''(t)\omega''(t)) = \begin{cases} 0 & \text{if} \quad \chi_{dh}(t) = 1 \\ R_{\text{dh}} & \text{Otherwise} \end{cases}, \tag{4.19}$$

where, the constant term, $R_{\text{dh}} = -0.15$.

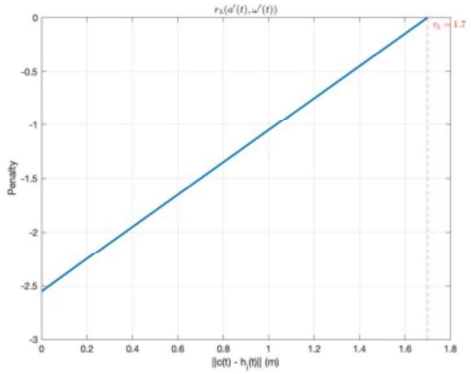


Figure 4.9. Reward function, $r_h(a''(t), \omega''(t))$.

The next three terms are activated only if the robot sees the human target. These aim to ensure robot's body and head move in a coordinated manner.

• The term r_{φ_H} introduces a penalty whenever the robot's optical center is not aligned with the target. In particular, if $f_H^g \in \mathcal{F}$ denotes the human target angle in spherical coordinates, let δf_H^g be defined as:

$$\delta f_H^g(t) = (f(t) - f_H^g(t))^T (f(t) - f_H^g(t)). \tag{4.20}$$

Hence, the term r_{φ_H} is defined as follows:

$$r_{\varphi_H}(a''(t), \omega''(t)) = \begin{cases} 0 & \text{if } \delta f_H^g(t) < \tau_H^g \\ R_{\varphi_H} \dot{\delta} f_H^g & \text{if } \dot{\delta} f_H^g < 0 \end{cases}. \tag{4.21}$$

In Equation 4.21 the constant terms are set as follows: $\tau_H^g = 0.175$ radians and $R_{\varphi_H} = 0.05$. The visualization of this penalty term is given in Figure 4.10.

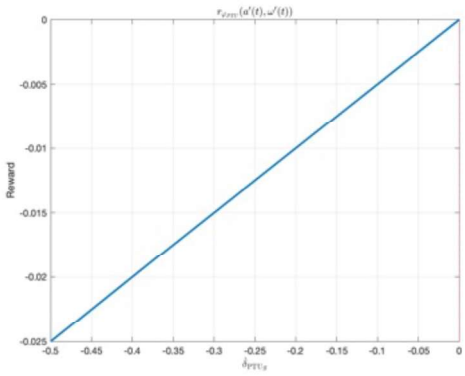


Figure 4.10. Reward function, $r_{\varphi_H}(a''(t), \omega''(t))$.

• The term $r_{h\alpha}$ introduces a reward based on the alignment of the robot's orientation with the human target. It is defined as:

$$r_{h\alpha}(a''(t), \omega''(t)) = r_{r\alpha}(a''(t), \omega''(t)) + r_{f_1\alpha}(a''(t), \omega''(t)). \tag{4.22}$$

The first term $r_{r\alpha}$ is concerned with the alignment of the body with respect to the target human.

$$r_{r\alpha}(a''(t), \omega''(t)) = \begin{cases} \left(1 - \frac{|e_{r\alpha}(t)|}{\tau_{r\alpha}}\right) & \text{if } \tau_{fw}^{-} \leq ||u_{rh}(t)|| < \tau_{fw}^{+} \text{ and} \\ & |e_{r\alpha}(t)| < \tau_{r\alpha} \text{ and } \theta_{fw} \leq \frac{\pi}{3} \\ R_{h\alpha} \cdot \tanh(|e_{r\alpha}(t)|) & \text{Otherwise} \end{cases} . \tag{4.23}$$

The second term $r_{f_1\alpha}$ is concerned head's alignment in the same manner:

$$r_{f_1\alpha}(a''(t), \omega(t)'') = \begin{cases} \left(1 - \frac{|e_{f_1\alpha}(t)|}{\tau_{f_1\alpha}}\right) & \text{if } \tau_{fw}^{-} \leq ||u_{rh}(t)|| < \tau_{fw}^{+} \text{ and} \\ & |e_{f_1\alpha}(t)| < \tau_{f_1\alpha} \text{ and } \theta_{fw} \leq 60^{\circ} \\ R_{h\alpha} \tanh(|e_{f_1\alpha}(t)|) & \text{Otherwise} \end{cases} . \tag{4.24}$$

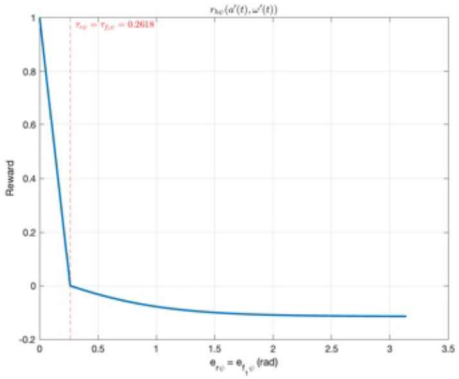


Figure 4.11. Reward function, $r_{h\alpha}(a''(t), \omega''(t))$.

The parameters in both Equation 4.23 and Equation 4.24 are set as $R_{h\alpha} = 0.15$ $\tau_{f_1\alpha} = \tau_{r\alpha} = 0.2618$ radians.

In Figure 4.11, same plot applies to its components, $r_{r\alpha}$ and $r_{f_1\alpha}$. The difference is that at the time of computation, they computed based on different parts of our robot body- either the base itself or the head.

• To align the robot's head with its body, we introduce a penalty function to minimize the pan angle. It aims to penalize the pan angle to be zero, which is the initial pose; that is, aligned with the robot's body at the start. Since the head mechanism will be moderated by the DRL algorithm (via action k_H) and not directly controlled, this will penalize the agent whenever its body diverges from the head angle. This penalty function is given as

$$r_{f_1}(a''(t), \omega''(t)) = R_{f_1} \frac{f_1(t)^2}{\pi^2}, \tag{4.25}$$

where, $R_{f_1} = -0.5$.

• We also give an additional fixed reward if the robot can stay consistently in the follow window for the last 12 steps out of 20, and we record this episode as successful. Similarly, if the robot is stuck due to the APF navigation method or loses the target

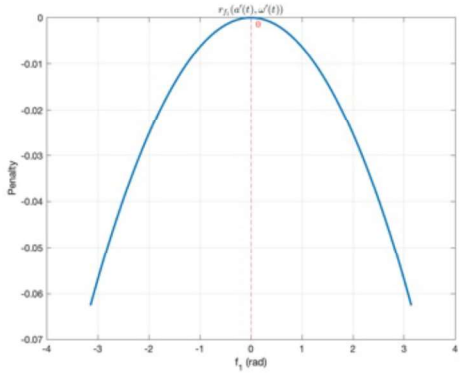


Figure 4.12. Reward function, $r_{f_1}(a''(t), \omega''(t))$.

human more than 20 steps, we penalize the agent for a fixed amount, record this episode as overtime, and reset the episode. These rewards are defined as $R_{success} = 40$ and $R_{overtime} = -5$ respectively.

4.4. Environmental Complexity and Learning

Previous work has introduced an environmental complexity measure for the training and test environments. We have altered the definition of environmental complexity slightly. The outer obstacle (β_0) inherently has room for its parameters, indicating that size information is included. Similarly, we introduced a term, κ, that contains the area of the room (w_r, l_r, width and length of the room, respectively). The complexity of the designed training and test rooms is computed using the plugin model data plugin; one can refer to Appendix G.1. The plugin provides enough information to extract the obstacle contents of the room and differentiate them based on our following complexity definitions. The upper side of Figure 4.13, where it says the "World model data" label, shows the visualization of an example room, entirely constructed using the plugin's information. Therefore, we can fit our elliptic representations and compute each room's complexity factor.

Consider a scenario with the set of obstacles, $\mathcal{O}$, with cardinality $N_o = |\mathcal{O}|$. The term $\beta_{comp}^{internal}$ given as

$$\beta_{comp}^{internal} = \frac{\big(N_o(N_o-1)\big)}{2(\sum_{i=0}^{N_o-1} \sum_{j=i+1}^{N_o} \beta_{ij}^{internal})} \quad . \tag{4.26}$$

The Equation 4.26 measures how close the obstacles are with respect to each other in combinations of two with the term $\beta_{ij}^{internal}$. It is given

$$\beta_{ij}^{internal} = \|o_i - o_j\|^2 - (\rho_i + \rho_j)^2, \quad \mathbf{i} \neq \mathbf{j} \neq 0. \tag{4.27}$$

The term β_{comp}^{outer} determines how close the obstacles are with respect to the center of the outer obstacle (i.e., the center of the room), defined as

$$\beta_{comp}^{outer} = \sum_{i=1}^{N_o} \beta_{i,comp}^{outer}. \tag{4.28}$$

In the Equation 4.28 the term $\beta_{i,comp}$ is defined as

$$\beta_{i,comp}^{outer} = \frac{(b_1 + \pi b_2)^2 - ||o_o - o_i||^2}{(b_1 + \pi b_2)^2}. \tag{4.29}$$

The size information of the obstacles relative to the overall dimension of the room we are currently missing from the complexity formula. Since we are interested in navigation, disregarding the height factor that can cause occlusions on the target human, we can simply take the ratio of the sum of internal obstacles' area to the outer obstacles' area given by

$$\eta = \frac{\sum_{i=1}^{N_o} \pi b_1 b_2}{\pi b_1 b_2}. \tag{4.30}$$

The static complexity can then be computed by

$$\mathrm{C_s} = \kappa \frac{\alpha_{comp}}{1 - \alpha_{comp}} (\eta \beta_{comp}^{internal} \beta_{comp}^{outer}). \tag{4.31}$$

The parameter κ is normalization factor of the room dimensions (w_r, l_r) relative to the robot, and defined as

$$\kappa = \frac{\max(w_r^2, l_r^2)}{\left(\rho_r\right)^2}. \tag{4.32}$$

Equation 4.31 has the term α_{comp}, and it is the weighing between the effect of the $\beta_{comp}^{internal}$ and the β_{comp}^{outer}. The dynamic complexity must also be computed to determine the dynamic obstacles' additional effect on the complexity. Those kinds of obstacles are mainly humans that are present in the environment, we will account for the ratio of their velocity to the robot's velocity to add their effects. The formula can be written as

$$\mathrm{C_d} = \mathrm{C_s} + \sum_{i=1}^{N_o} \frac{v_i}{v_r}. \tag{4.33}$$

This should capture the effects on the complexity, as the number of dynamic obstacles and their velocities increase, so does the dynamic complexity.

4.5. Simulation Setup for Training

Simulation setup for training follows an episodic learning framework, where rewards are assigned at each step based on the robot's actions. Figure 4.13 presents the overall architecture.

First, an environment is set up for the training. Since this time, we have altered the obstacle generator, which finds the internal obstacles more accurately, including walls. We also introduced dynamic obstacles that were treated as human entities. This design choice was necessary because the low-polygon human meshes in the Gazebo simulation lacked sufficient visual distinction, making it impractical to visually identify the target human based on appearance alone.

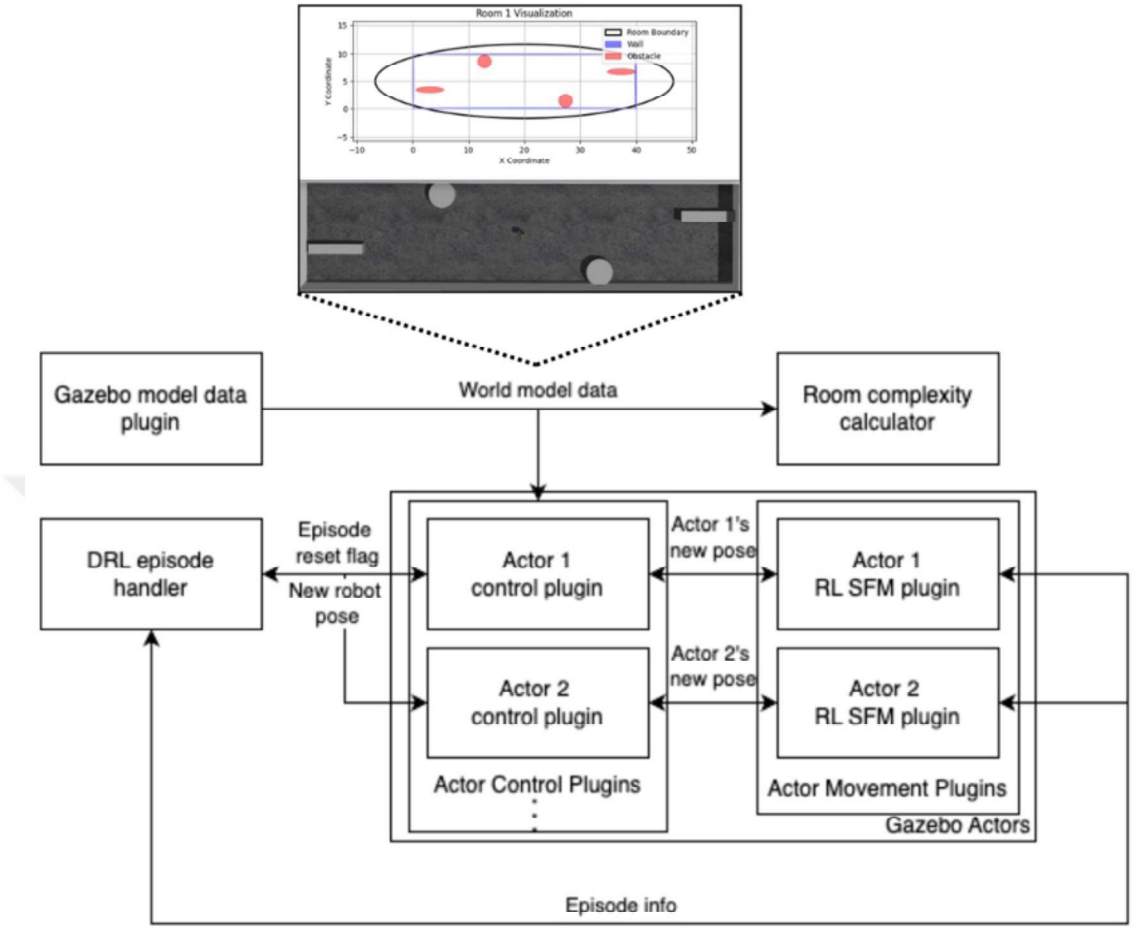


Figure 4.13. DRL and Gazebo plugin interaction pipeline.

Next, the world model data is constructed. Unfortunately, it was not feasible to directly reuse the software developed in previous work. This limitation arises from differences in the simulation environment: the current work employs Gazebo 11, which does not provide object size information through its standard ROS interface. Instead, it only outputs precise positional data for models and their associated links. To overcome this limitation, we developed a custom Gazebo-ROS plugin capable of extracting and publishing additional model attributes, including position, size, orientation, and velocity (for non-stationary models or links). This enhanced interface enables real-time computation of both the static and dynamic complexity of the environment. As a result, it becomes possible to construct diverse and increasingly complex scenarios on the fly, without the need to manually design and export them beforehand. This

represents a significant departure from the training setup that had been developed for Social APF-RL navigation, where environment complexity was pre-defined, and static scenarios were explicitly modeled and then exported to the simulation platform. This plugin is detailed in Appendix G.1.

We utilized this Gazebo data plugin to determine suitable spawn locations, ensuring all physical constraints in the environment were respected. We have implemented a sophisticated entity respawning mechanism at the start of each episode, specifically for both the robot and the target human. This ensures that while the initial condition of visibility is met, the environment remains diverse and unpredictable. Incorporating this controlled randomness is critical to prevent overfitting to specific initial states and to encourage robust policy learning that generalizes across varied scenarios.

Spawn locations for both the robot and actors are sampled dynamically based on the current free space available in the environment. For realistic pedestrian movements, we integrated the Gazebo SFM plugin [47], which employs the Social Force Model (SFM) to guide actor navigation. The SFM-based navigation allows actors to adapt their motion based on surrounding obstacles and is highly customizable (see Appendix C for details). We extended this SFM plugin by incorporating additional logic to support dynamic respawning of actors and to allow them to continue following predefined waypoints using the SFM planner. Furthermore, the extended plugin tracks actor state information within each episode, enabling us to define reset conditions based on actor behavior and environmental interactions. This plugin is detailed in and Appendix G.3.

In order to dynamically respawn actor models, we developed a dedicated actor control simulation plugin. The actor control simulation plugin interfaces with the enhanced SFM plugin to manage actor respawns and to compute appropriate robot spawn positions, including yaw orientation, so that the robot initially faces the target human at the start of each episode. This setup facilitates data collection from a diverse range of configurations within the same room, thereby increasing sample variety for

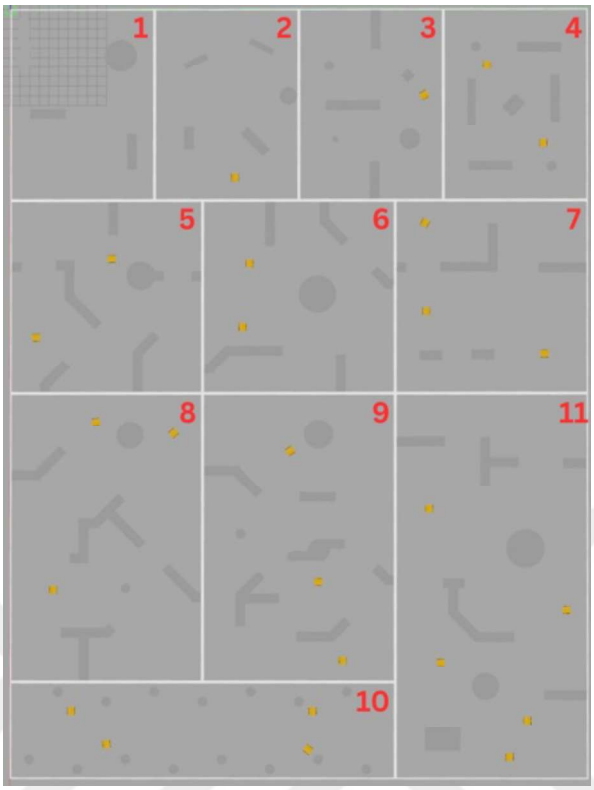


Figure 4.14. Training is done considering eleven different environments varying in environmental complexity.

the deep reinforcement learning (DRL) model. This plugin is detailed in Appendix G.2. Hence, each target human is managed by a paired instance of these plugins, and a centralized DRL respawn handler coordinates with them to manage episode initialization and transitions throughout the training process.

4.6. Learning Body-Head Coordination

An extensive training process is carried out for the Head-Coordinated Social APF-RL framework for human following. To support this, eleven distinct environments—illustrated in Figure 4.14—have been designed with progressively increasing complexity, as detailed in Table 4.2. All rooms have one main actor, the target human to be followed. Then, we add dynamic obstacles that are treated as human entities as

Table 4.2. Head-Coordinated Social APF-RL for Human Following: Environmental complexities of training rooms.

Room #	C_s	C_d
1	8.82	9.15
2	11.82	12.22
3	13.04	13.51
4	14.09	14.63
5	12.34	12.94
6	9.56	10.23
7	8.32	9.06
8	27.67	28.32
9	33.65	34.49
10	22.51	23.41
11	19.99	20.86

explained previously. Their movements are governed by SFM.

- Rooms two and three have one additional human;
- Rooms four, five, and six have two additional people;
- Rooms seven, eight, and nine have three additional people;
- Room ten has four additional humans;
- Room 11 has five additional humans.

For each room, we increased the speed of the target human by 0.05 m/s, starting from 0.5. The additional humans have 0.8 m/s for rooms 2 to 5, 1.2 m/s for rooms six and seven, and the remaining rooms that have additional humans have 1.0 m/s as their speed. For initial training, we spawned the robot behind the target human whenever it was possible to grasp the following behavior first, then went on to full randomness to create a variety of samples to learn from.

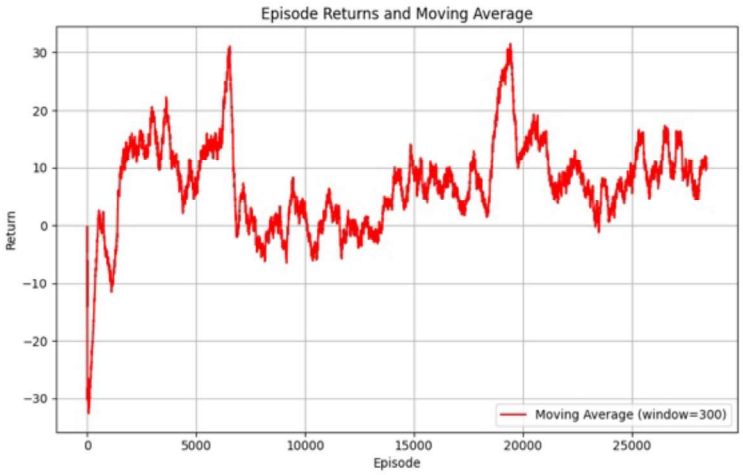
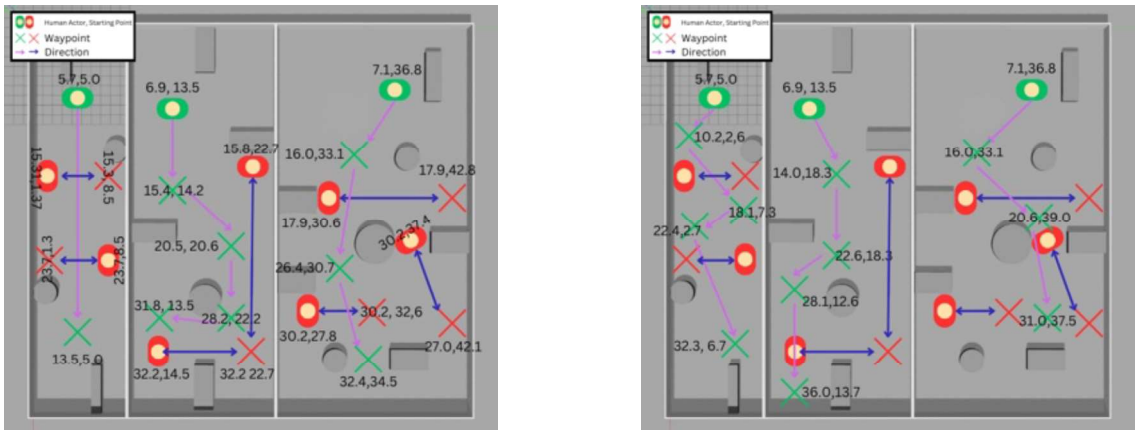


Figure 4.15. Episode rewards over 950000 iterations.

At the beginning of each episode, we assume that the target human is initially positioned within the robot's field of view. However, the subsequent trajectory of the target human is not constrained to move away from the robot. Consequently, the robot must learn to continuously reposition itself to maintain visual contact, regardless of the human's movements.

The training process is conducted in multiple stages, following a curriculum learning approach. Initially, the focus is on teaching the robot the fundamental human-following behavior.A transition to a more complex room is triggered when at least 80% of the last 20 episodes are deemed successful, defined by the robot maintaining its position within a predefined follow window for the majority of the episode duration. Once a stable policy is achieved, we progressively adjust the reward function weights to emphasize coordinated body-head motion. This staged training allows the agent to first master basic navigation and then gradually incorporate more complex coordination behaviors. Noting that we have used our revised obstacle generator and new permissible APF navigation module during the training. Lastly, the training is done using an Intel i9-13900K processor with a NVIDIA GeForce RTX 3060 TI 8GB VRAM graphics processor unit.



(a) (b)

Figure 4.16. Body-head coordination test environments, (a) first trajectories, (b) second trajectories for each room.

Figure 4.15 illustrates the evolution of the total reward throughout training, visualized using a moving window average. Notable transitions in environment complexity—specifically, from Room 1 to Room 2 and from Room 5 to Room 7—are evident in the reward progression, with distinct peaks occurring around a reward value of 30.As the policy is learned from scratch and the initial rooms (Rooms 1–4) are relatively small, Room 2 was found to be overly constrained for effective learning. Despite running approximately 5,000 episodes in Room 2, the robot failed to demonstrate adequate learning progress. Consequently, we skipped to Room 5, which offers more spatial flexibility and proved more conducive to learning the human-following task.

4.7. Experimental Results

The performance of learning is evaluated considering three different rooms as shown in Figure 4.16. Table 4.3 details the complexity levels of the test environments. These complexities are notably higher than those seen during training, and the trained model has never encountered these specific configurations. As the complexity measures are based on static snapshots of the environments, and the initial actor and robot poses are shared across setups, the complexity values remain the same across

trajectory variants within each room. Each room includes two distinct target human trajectories, resulting in six unique test configurations. The first and second trajectory setups are illustrated in 4.16(a) and 4.16(b) respectively. Three different approaches are considered:

- Social APF-RL Navigation - Head is stationary. This is considered as a baseline approach.
- Social APF-RL for Human Following - Head moves, but there is no body-head coordination. This enables us to study the effect of head movement without any body-head coordination.
- Head-Coordinated Social APF-RL for Human Following - There is body-head coordination. The evaluations are conducted using our most recent trained model, which has completed 950,000 training iterations.

During these tests, we have used our Social APF-RL trained with the new APF and obstacle generator algorithm. For each approach, tests are repeated in each room five times with different initial poses of the robot. Altogether, there are 30 runs for each approach.

In the current phase, the evaluation primarily focuses on the human-following performance, with coordination metrics considered as secondary objectives. Performance is measured with respect to the following criteria:

- Success %: The success is defined as whether the human target is being able to followed or not.
- Social violation %: Since we are interested in social compliance, if any human's personal space is violated, we mark that result as a social violation, which is coming closer than 1.2m by the rules of proxemics.
- Failure %: This occurs if either the robot loses the human target or collides with an object.
- Avg. Dist. (m): Average of the distances measured between each human and the

Table 4.3. Head-Coordinated APF-RL for Human Following: test room environmental complexities.

Room #	C_s	C_D
1	15.24	17,31
2	22.38	23.78
3	31.29	34,09

robot.

- Avg. Min. Dist. (m): Average of the minimum distance recorded between each human and the robot.
- Avg. pan angle: Average pan angle of the robot's head. Ideally, it should be close to zero, meaning that the head is aligned with the body.
- Avg. human align. Angle: Average angle between the robot body and the head combination and the human target. Ideally, it should be close to zero, meaning that the robot is aligned with the human target.

Table 4.4 presents comparative results. It is important to note that all metrics were recorded continuously throughout the simulation, not only during instances of human detection. First, it must be noted that social violations for Social APF-RL algorithm retained low percentage of violations as before, considering now we have a non-static goal and using our revised obstacle generator with the new permissible APF module that the model was not trained on. We also presented the tests for not using the head mechanism as a baseline, and clearly it performs poorly compared to using the head mechanism. For the proposed head-coordinated approach, the training of the model needs to be continued. Although the difference is minimal, this approach increases the violation of human personal space, which may be due to the lack of training in human-populated environments. We have to make sure the learning is generalized by the observations, and then go on to tuning the coordination behavior itself. Currently, we have penalty mechanisms that are overwhelmed by the following goal; therefore, the performance is worse off compared to Social APF-RL for human

following without body-head coordination. The average pan angle is increased while the overall angle with respect to human alignment has decreased with the proposed approach. We are to tune the weight of reward functions in the future of the learning to enforce better coordination behavior. Therefore, we intend to achieve such a model that can avoid static or dynamic obstacles and still be able to align itself as soon as possible towards the human target.

Table 4.4. Comparative performance results: Social APF-RL Navigation, Social APF-RL for Human Following and Head-Coordinated Social APF-RL for Human Following.

Method	Room#	Success %	Social Violation %	Failure %	Elapsed time (s)	Avg. Dist. (m)	Avg. Min. Dist. (m)	Travelled Dist. (%)	Avg. pan Angle (°)	Avg. human align. (°)
Social APF-RL Navigation	Room1	70%	10%	20%	48.6430	6.5462	1.5783	145	-	122.9562
	Room2	50%	0%	50%	27.0659	4.9670	1.5742	116	-	123.081
	Room3	30%	0%	70%	54.8625	10.1317	3.0039	124	-	92.4132
Social APF-RL for Human Following	Room1	90%	10%	0%	47.6051	6.5281	1.6193	145	34.4108	123.8193
	Room2	90%	10%	0%	68.8564	9.3593	2.1373	108	35.0582	127.0267
	Room3	100%	0%	0%	54.3166	10.0341	3.0043	114	38.2891	90.6068
Head Coordinated Social APF-RL for Human Following	Room1	60%	20%	30%	60.8557	7.0878	1.6953	211	46.7716	88.5793
	Room2	60%	10%	30%	70.9818	10.4770	3.3538	123	50.9627	96.1160
	Room3	30%	0%	70%	33.1168	4.8974	1.2384	168	52.5435	81.0173

5. ENVIRONMENTAL SENSING FOR RECOVERY FROM TARGET LOSS

This chapter focuses on environmental sensing, which is essential for enabling recovery from the loss of a human target—an indispensable aspect of reliable human following. To ensure continuous following, the robot must be equipped with mechanisms that allow it to consistently track the human target. As it navigates, the robot monitors all relevant objects, including the human and surrounding obstacles. In the event of a sudden loss of the target, the robot can leverage its tracking history to infer the human's potential whereabouts. Such loss may result from sensing failures or from the human becoming temporarily obstructed, for example, when passing through architectural features like doors. The latter deals with structural challenges in the environment that inherently affect visibility. By detecting and reasoning about such spatial obstructions, the robot can make informed decisions to handle these interruptions effectively and resume following behavior.

Doors, in particular, are a common architectural feature that can temporarily obstruct the robot's line of sight, potentially leading to target loss [48]. While most doors share similar low-level geometric features—typically rectangular in shape—they also exhibit diverse high-level characteristics. These include differences in functionality, such as flip or sliding mechanisms, varying movement constraints, and configurations involving single or multiple doors. Door detection presents a significant challenge due to the wide variability in door types, sizes, colors, and operational states (e.g., closed, partially open, or fully open). For instance, a single-leaf door is a flip-style door that rotates around a single pivot axis relative to its frame.

Importantly, recognizing a door as an obstruction is not sufficient for the robot to determine whether a human can pass through it—or whether it can follow. Effective human following in such scenarios requires the robot not only to detect doors but also to infer their current status (e.g., open, closed, partially open) and decide accordingly.

The status of a door is determined by its orientation angle: a door is considered closed if its angle is between 0° and 10°, partially open if the angle ranges from 10° to 70°, and open if the angle exceeds 70°. This capability is essential for deciding whether the robot should attempt to pass through a doorway or seek alternative recovery strategies when the human target is lost.

The contributions of this chapter can be summarized as follows:

- Mapping and reasoning for the recovery of a followed, but lost, human target.
- Door and door status detection
- A RGB-D door dataset consisting of 1506 samples.

The outline of the chapter is as follows: Related literature is discussed in Section 5.1. Following, we have discussed our tracking and mapping mechanism in Section 5.2. Then, door detection is explained in Section 5.3. Afterwards, door status and orientation detection are explained in Section 5.4. We finally bring all together in Section 5.5, and lastly, to finalize this chapter, we present our experimental results in Section 5.6.

5.1. Related Work

Our review of related work focuses on two key aspects of the problem:

(i) Target recovery following loss of the human during navigation;
(ii) Reasoning about doors — that can obstruct tracking and affect the robot's ability to maintain visual contact with the target.

5.1.1. Recovery from Human Target Loss

Human following approaches must also address recovery from human target loss during navigation. In many existing works, a tracker module is employed to handle

short-duration losses. However, when the tracker fails, explicit recovery procedures become necessary. For example, in [49], a re-identification module uses previously learned features such as the target's face and torso to identify potential candidates obtained through a 360-degree scan. If the re-identification fails, the robot switches to a search mode.

In another approach [50], recovery is based on the assumption that target loss is likely caused by the target turning around corners. The robot navigates to the last known position of the human and initiates a detection routine by rotating in place to search for the lower part of the human body. If the human is still not found, the robot updates a Kalman Filter to estimate the human's position and selects a new search location, potentially guided by an existing map.

An alternative strategy involves detecting events that may lead to human target loss as they occur - enabling the robot to respond proactively. For example, if a door transient event and pedestrian activity such as passing through a door - can be detected, this contextual information can assist in maintaining following behaviour. Door transitions are detected various methods including a random forest classifier, CNN, and LSTM networks.

5.1.2. Obstruction Detection and Doors

This work primarily focuses on doors as the main source of visual obstruction during human following. A variety of door detection approaches have been proposed - ranging from traditional algorithmic methods to deep-learning based techniques.

Algorithmic approaches often rely on geometric reasoning and plane fitting methods such as RANSAC. For example, the method presented in [51] leverages structural information from doors by extracting door components from point cloud data. It utilizes the PCL library's region growing algorithm based on surface normals to segment the data. Detected door candidates are then rated using a Gaussian probability mea-

sure to assing a confidence score indicating the likelihood of the object being a door. The algorithm operates within a processing time of 0.5 to 2 seconds, depending on the robot's distance to the door. In another approach, doors are identified by fitting a four-sided polygon to edge features extracted from RGB images [52]. Similarly, in [53], data from both a laser sensor and an RGB camera are fused to detect doors while navigating corridors. This method relies on vertical line fitting and geometric constraints. More complex sensor configurations have also been explored: a mobile robot equipped with a 3D LiDAR, a 2D laser scanner, and a color camera detects both open and closed doors by identifying occupied voxels in 3D space and fitting rectangular planes to represent the door surfaces [54, 55].

More recent work have proposed using deep learning networks for door and status detection. A single-shot multibox door detector network trained on a dataset containing door and door handle classes is presented in [48]. Door status is inferred from depth variations. Subsequently, the RANSAC plane fitting algorithm is applied to identify door planes and, importantly, their normals. A humanoid robot with a monocular camera to detect a door and its orientation using a CNN-based detector [56]. A YOLO based detection network is trained to classify handle, cabinet door, room door, and refrigerator doors [57]. In [58], three different door state classification methods - each using different type of information, have been presented. In the first method, RGB and depth data are used with FastFCN or FC-HarDNet for segmentation. A point cloud is generated from depth data using Open3D, and PointNet is applied to classify the door's state. The second method relies solely on PointNet for 3D classification, using only depth data to construct the point cloud via the Open3D library, enabling faster processing. The third method employs purely 2D door detection and state classification using only RGB data. DetectNet, SegNet, and BiSeNet are used for door detection, followed by AlexNet and GoogleNet for door state classification. The methods were tested on the Jetson Nano, with the second method delivering an inference speed of approximately 6 FPS. A transformer network (Detection Transformer with ResNet-50) based on a CNN backbone that has pre-trained weights has been trained [59].

5.2. Tracking Objects of Interest

The robot must track all objects of interest, including the human target. The tracking system fuses data from odometry, the RGB-D camera, and multiple detector inputs. It uses a Kalman filter operating within the fixed odometry frame to compensate for the robot's movement.

As the robot navigates, it continuously tracks all encountered obstacles, including the human target. Each obstacle is tracked based on its centroid. For human detections, the depth values around the bounding box center are averaged and projected into the odometry frame to obtain the centroid measurement. The bounding box is either taken from incoming detections or, if unavailable, inferred from existing tracks. New tracks are created for obstacles detected for the first time.

For each incoming centroid measurement o_i, data association is performed by computing the Euclidean distance between the predicted track position and the measurement. Two gating thresholds are applied: a strict threshold for new and unclassified objects and a more relaxed threshold for human tracks. The update process follows two conditions - if the centroid falls within a track's gating threshold, the track is updated using Kalman filter correction step; otherwise a new track is created. The Kalman filter has a four-dimensional state vector that is defined as

$$\mathbf{x} = \begin{bmatrix} x_1 & x_2 & v_1 & v_2 \end{bmatrix}^T. \tag{5.1}$$

The states given in Equation 5.1 are:

- $x = (x_1, x_2)$ are the planar positions of the track in the odometry frame.
- $v = (v_1, v_2)$ are the planar velocities of the track in the odometry frame.

Suppose state noise covariance matrix $\mathbf{Q}$ is known. Measurement noise covariance matrix $\mathbf{R}$ is also known. Let $\mathbf{R}_{\text{RS-LIDAR-16}}$ denote the one associated with the LIDAR

sensor as obtained from its datasheet:

$$\mathbf{R}_{\text{RS-LIDAR-16}} = \begin{bmatrix} 0.0004 & 0 \\ 0 & 0.0004 \end{bmatrix}. \tag{5.2}$$

Similarly, the ZED2 stereo camera's noise covariance matrix for two different confidence levels based on its depth readings. Those are based on the 2 and 12 meters and defined as follows

$$\mathbf{R}_{\text{ZED2, 2m}} = \begin{bmatrix} 0.000256 & 0 \\ 0 & 0.000256 \end{bmatrix}, \quad \mathbf{R}_{\text{ZED2, 12m}} = \begin{bmatrix} 0.2304 & 0 \\ 0 & 0.2304 \end{bmatrix}. \tag{5.3}$$

Assuming velocity to be constant, let the state transition matrix A be defined as:

$$A = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{5.4}$$

where Δt term encodes the elapsed time since the last update. Furthermore, let $\mathbf{H}$ be observation matrix:

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}. \tag{5.5}$$

The next prediction state equation is:

$$\hat{\mathbf{x}}_{k|k-1} = A\hat{\mathbf{x}}_{k-1|k-1}. \tag{5.6}$$

Error covariance matrix $\mathbf{P}$ is also updated:

$$\mathbf{P}_{k|k-1} = A_{KF}\mathbf{P}_{k-1|k-1}A_{KF}^T + \mathbf{Q}. \tag{5.7}$$

With the new measurement $\mathbf{z}_k$, the state is updated as:

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k \left(\mathbf{z}_k - \mathbf{H}\hat{\mathbf{x}}_{k|k-1} \right). \tag{5.8}$$

Here, the matrix $\mathbf{K}_k$ is the Kalman gain:

$$\mathbf{K}_k = \mathbf{P}_{k|k-1}\mathbf{H}^T \left(\mathbf{H}\mathbf{P}_{k|k-1}\mathbf{H}^T + \mathbf{R} \right)^{-1}. \tag{5.9}$$

After the correction update, the error covariance, $\mathbf{P}$, is updated by

$$\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k\mathbf{H})\,\mathbf{P}_{k|k-1}. \tag{5.10}$$

All measurements are processed in the odometry frame to compensate for the robot's movement - although odometry transformation itself can accumulate errors over time.

5.3. Door Detection

As discussed previously, the main motivation for door detection is to enable the robot to identify potential passages within the environment. This becomes especially important when the robot loses sight of the human target. In such cases, door detections can be used to infer whehther the target may have passed through a doorway. The robot can then navigate to the detected door in an attempot to reacquire the target and resume its human following task.

5.3.1. Learning Datasets

The datasets used for the training are listed in the Table 5.1. We also have captured an in-house dataset at Boğaziçi University, North Campus, Square Block's second and fourth floor, using a ZED 2 stereo camera, capturing RGB and depth images and hand-labeling them. Our dataset is captured from a camera lower than the robot's, but captures frontal views of doors at various angles. This specific dataset

Table 5.1. List of door image datasets.

Dataset	**Depth**	**RGB**	**Door Samples**	**Training**	**Test**	**Is labelled**
DeepDoors 2 [58]	✓	✓	2998	✓	×	✓
Door Detect [57]	×	✓	386	✓	×	✓
Door Detection-v3[1] [61]	×	✓	4493	✓	×	✓
Doors-v1[1] [62]	×	✓	6958	✓	×	✓
DWSonder[1] [60]	×	✓	3140	×	✓	✓
ADE20K [63]	×	✓	4672	–	–	✓
Multi-Env. Obj. Det. [64]	×	✓	1873	–	–	✓
SRIN Doorway [65]	×	✓	14885	–	–	×
Ours	✓	✓	1506	×	✓	✓

is reserved for testing purposes, for initial deployment will be around places where we capture those images. The captured novel dataset consists of 1506 images, 703 of which directly capture a flip-door with different orientations and perspectives. The remaining 803 images are extracted from the corridor human following scenario. We also used the DWSonder [60] dataset that consists of 3140 door images (with an additional 1900 images for negative data that consist of windows) for testing. Datasets that we leave as − can be used to expand the training set for further training.

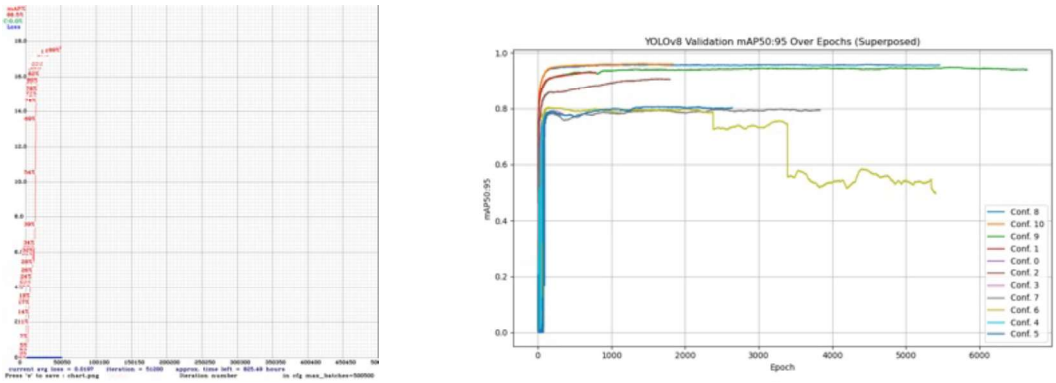
5.3.2. Door Detection Learning

Door detection is learned using two different architectures:

- YOLOv4 [66] (with CSP-X-Swish configuration);
- YOLOv8 [67](Medium architecture).

The training is done using an Intel i9-13900K processor and NVIDIA RTX 3060 TI 8GB VRAM GPU.

[1]Dataset has augmented data.



(a) (b)

Figure 5.1. Trained (a) YOLOv4 and (b) YOLOv8 (all configurations superposed) door detectors – validation mAP50:95 over epochs.

The YOLOv4 model with CSP-X-Swish configuration and pre-trained weights are fine-tuned with the datasets specified in Table 5.1. Input image sizes are 320x320 as resolution with a batch size of 128 and subdivision of 64. The training process incorporates Stochastic Gradient Descent (SGD) as an optimizer, starting with a learning rate of 0.001 and reducing it as training goes on. The specific configuration used is utilizing architectural improvements CPSNet [68] with Swish activation functions for improving feature propagation and detection performance. The YOLOv4 training is stopped once the model reaches mAP50:95 of 88.5% on the validation dataset. One can see that the training process is terminated in various epochs. We only saved 20 weights for YOLOv4 with a CPS-X-Swish network.

The YOLOv8 medium model is taken and fine-tuned similarly to YOLOv4 training. A total of 11 different training argument configurations are tried for training. See Appendix D. The YOLOv8 training is done with 11 different argument configurations, each stopped at varying mAP50:95 values. Those training evaluation curves are given in Figure 5.1(a) and Figure 5.1(b) for YOLOv4 and YOLOv8, respectively. There are 307 weights saved (with varying argument configurations) for the YOLOv8 medium network.

5.4. Detecting Door Status and Orientation

To recover the human target after loss — particularly when the target passes through a door—the robot must determine the door's status (open, closed, or partially open) to decide whether it can proceed. If the door is not fully open, the robot also needs to estimate the door's orientation. The proposed pipeline for door status and orientation detection is illustrated in Figure 5.2.

Once a door is detected, regions of interest (ROIs) are extracted with slight enlargement to account for possible localization errors from the YOLO detector and inaccuracies in detecting the door frame. Let $\mathcal{D}$ denote the corresponding set of pixels. For each pixel $p = (p_1, p_2) \in \mathcal{D}$, let $d(p)$ encode the corresponding depth value. Assuming camera with intrinsic parameters f_1, f_2, o_1, o_2, a 3D point cloud model $\mathcal{D}_P$ of the door is constructed using depth data. For each $p \in \mathcal{D}$, the corresponding 3D point $P \in \mathcal{D}_P$ is computed as follows:

$$P = \begin{bmatrix} P_1 \\ P_2 \\ P_3 \end{bmatrix} = \begin{bmatrix} \frac{(p_1 - o_1)P_3}{f_1} \\ \frac{(p_2 - o_2)\cdot P_3}{f_2} \\ \frac{d(p_1,p_2)}{\sqrt{\left(\frac{p_1-o_1}{f_1}\right)^2 + \left(\frac{p_2-o_2}{f_2}\right)^2 + 1}} \end{bmatrix}. \tag{5.11}$$

This is done considering the current height of the camera, h_t, accounting for the tilt angle (effective height), which can alter the height of the camera position with

$$h_t = h_b \cos(f_2), \tag{5.12}$$

where, h_b is the height of the camera when the PTU is at its origin position. Lastly, we apply the random sample consensus (RANSAC) method to extract planar surfaces - primarily doors and frame planes - in order to estimate the door's orientation. The criteria for determining RANSAC inliers are defined by:

$$d_{RANSAC} = \frac{|\zeta_1 P_1 + \zeta_2 P_2 + \zeta_3 P_3 + \zeta_4|}{\sqrt{\zeta_1^2 + \zeta_2^2 + \zeta_3^2}} \leq \tau_{RANSAC}. \tag{5.13}$$

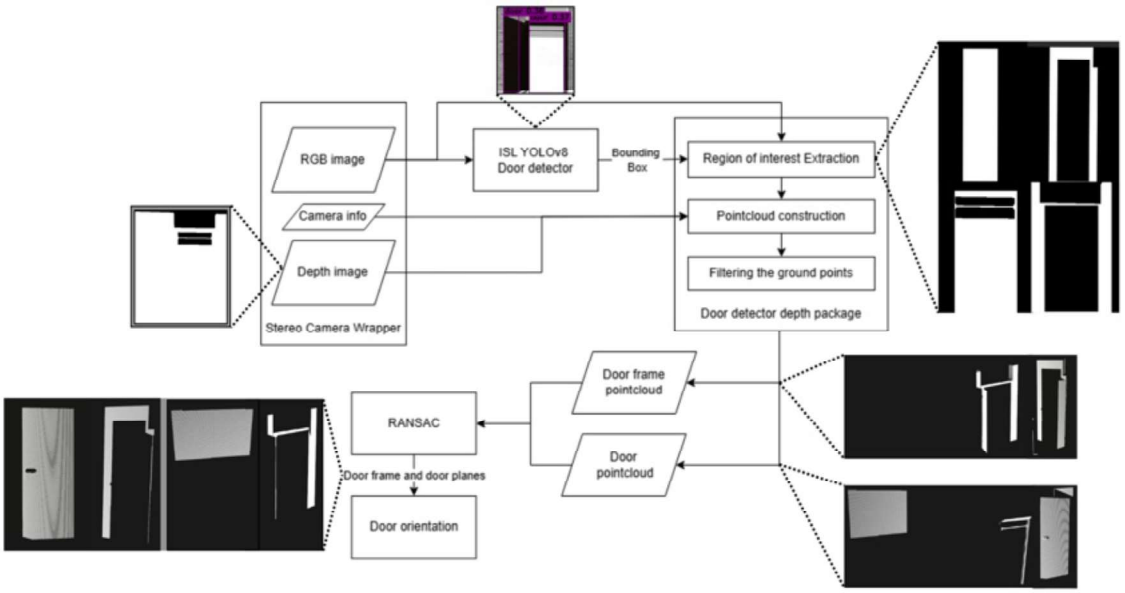


Figure 5.2. Door status detection pipeline.

The terms ζ_i, $i \in \{1, 2, 3, 4\}$, in Equation 5.13 are generic plane parameters and τ_{RANSAC} is the threshold for inliers. The plane obtained from RANSAC inliers provides an average depth estimate and can be used to infer the door's status. However, this approach is not always reliable - particularly in cases where an open door is not directly visible due to occlusion from the robot's viewpoint, making orientation estimation difficult. In such cases, analyzing the plane equations of both the door and the door frame can offer useful insights. If the depth difference along the optical axis between these planes exceeds typical door thickness dimensions, we can confidently infer that the door is open. Otherwise, more careful reasoning based on the orientation and relative positioning of the planes is required to determine the door status.

To determine the orientation of the door relative to its frame, two alternative methods have been considered:

- DO1: Plane normal comparison;
- DO2: Comparison of door normal and door pivot axis

In the first method, the normal $n_{door} \in \mathbb{R}^3$ of the detected door plane and the normal $n_{frame} \in \mathbb{R}^3$ of the door frame plane are extracted using RANSAC. The angular difference θ_{door_i} between these normals provides an estimate of the door's rotation and thus its opening angle:

$$\theta_{door_i} = \arctan\left(\frac{\|n_{\text{door}} \times n_{\text{frame}}\|}{n_{door} \cdot n_{\text{frame}}}\right). \tag{5.14}$$

In the second method, the door pivot axis $n_{pivot} \in \mathbb{R}^3$ and its position are first determined. The former can be estimated as follows:

$$n_{pivot} = \frac{n_{door} \times n_{frame}}{\|n_{door} \times n_{frame}\|}. \tag{5.15}$$

Its location can be found through the intersection of door plane with the frame plane. For this, we treat one coordinate as a free variable. In particular, the free variable is selected based on the largest component of the n_{pivot}. It is then set to zero. Then, we construct a 2x2 matrix using the normals for solving for the remaining components, which will indicate the pivot point denoted as P_0. Assuming the z-component to be the biggest component of the pivot axis, then we would have the following

$$\begin{pmatrix} n_{door,1} & n_{door,2} \\ n_{frame,1} & n_{frame,2} \end{pmatrix} q = \begin{pmatrix} n_{door,1} & n_{door,2} \\ n_{frame,1} & n_{frame,2} \end{pmatrix} \begin{pmatrix} q_1 \\ q_2 \end{pmatrix} = \begin{pmatrix} -\zeta_{4_{door}} \\ -\zeta_{4_{frame}} \end{pmatrix}. \tag{5.16}$$

Here, the parameters $\zeta_{4_{door}}$ and $\zeta_{4_{frame}}$ designate distance of the respective planes from the origin. The vector q can be solved using the QR decomposition to find the pivoting point. Once we have the pivot point q, letting

$$P^0 = \begin{bmatrix} q \\ 0 \end{bmatrix}. \tag{5.17}$$

We define the covariance matrix of the point cloud around it as:

$$\Sigma = \frac{1}{\mid \mathcal{D}_P \mid} \sum_{P \in \mathcal{D}_P} (P - P^0)(P - P^0)^T. \tag{5.18}$$

Eigen decomposition of the covariance matrix gives the eigenvector corresponding to the smallest eigenvalue as an estimation of the door normal $\hat{n}_{door}$. The door rotates around the hinge, so we are interested in rotation in the plane that is perpendicular to the pivot axis $\hat{p}$. To remove any component along the pivot axis, we have to project the door normal to the plane orthogonal to the pivot axis, which can be done with

$$\hat{n}'_{door} = \hat{n}_{door} - (\hat{n}_{door} \cdot n_{pivot}) n_{pivot}, \tag{5.19}$$

where, $\hat{n}'_{door}$ is the estimated door normal. This will ensure we do not have any component on the pivot axis. Then, we can use a similar equation given in Equation 5.14 using our door normal estimate and frame normal we already have at hand to come up with the door orientation.

Once the angle θ_{door} is estimated, the door status can be determined considering its magnitude. We denote four different statuses for a door. Their conditions are listed below:

- Closed: $0 \leq \theta_{door} \leq 15$,
- Semi-opened: $15 < \theta_{door} \leq 60$,
- Opened: $60 < \theta_{door} \leq 180$,
- Unknown: if there is no recent update on the door or the algorithm handles an exception.

5.5. Target Loss and Door Finding

To identify which door may have caused the loss of the human target, we project all door tracks onto a common 2D plane within the odometry frame. A KD-Tree is

constructed from these projected door positions to enable efficient spatial querying. When a human track is lost, we query the KD-Tree using the last known position of the target to find the nearest door(s) within a predefined distance threshold. If the last known position of the track overlaps with a door track, the human is labelled as "lost due to obstruction". The associated door's information is recorded for further analysis, which can then be used to recover the human target.

5.6. Experimental Results

Experimental evaluation is conducted in two stages:

- First, door detection performance is evaluated.
- Next, door-orientation detection performance is studied.

5.6.1. Door Detection Performance

Detection performance of the trained networks is evaluated using the following metrics:

- Mean average precision calculated based on intersection over union (IoU) thresholds ranging from 0.5 to 0.95 (mAP50:95);
- Average precision (AP);
- F1-score;
- False positives per image (FPPI);
- IoU mean;
- IoU standard deviation;
- Expected calibration error (ECE);
- Average inference time.

We have evaluated the weights with different test image sets separately. The detailed door evaluation results are given in Appendix D. The overall, aggregated results of

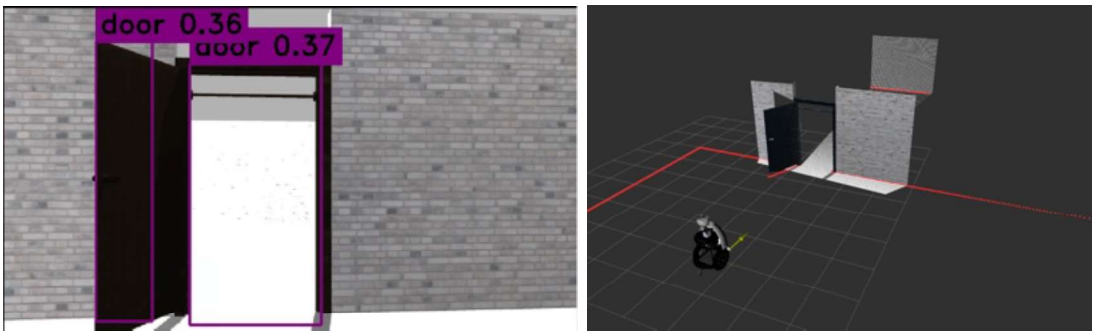
Table 5.2. Door detector evaluation: aggregated results, top 5 weights by mAP.

Weight number	mAP50:95	AP50	AP75	AP	F1	FPPI	IoU mean	IoU std.	ECE	Avg. Inf. Time
v8_8_3500	0.51	0.73	0.53	0.16	0.14	6.80	0.80	0.09	0.38	0.00494
v8_10_1700	0.50	0.73	0.54	0.17	0.16	7.13	0.81	0.10	0.37	0.01104
v8_8_3600	0.50	0.71	0.51	0.16	0.14	7.14	0.80	0.10	0.39	0.00495
v8_10_1800	0.49	0.73	0.48	0.17	0.16	6.64	0.80	0.10	0.35	0.01105
v8_10_last	0.48	0.72	0.49	0.17	0.16	6.40	0.80	0.10	0.36	0.01104

test image sets sorted with respect to mAP50:95 and top 5 weights are presented in Table 5.2. We evaluated the trained weights using separate test image sets separately. Detailed results of the door detection evaluation are provided in Appendix D. The aggregated results across all test sets - sorted with respect to mAP50:95 - and top five performing weight combinations are presented in Table 5.2. Note that the first column, 'Weight Number' encodes the YOLO version, followed by the training configuration number (specific to YOLOv8), and the epoch number—each separated by an underscore.

The results show that YOLOv8 weights and specifically argument configuration numbers 8 and 10 dominated from the mAP50:95 metric. The YOLOv4-CSP-X-Swish model's mAP50:95 stuck around 0.39, while YOLOv8, configuration number 8, reaches 0.51. The highest mAP yield by the YOLOv8 with configuration 8 at the 3500th epoch. Configuration 8 has input image sizes of 640x640 with default augmentations, a batch size of 12, a nominal batch size of 128, RMSProp as the optimizer, and cosine learning rate decay starting from 0.001, decaying to 0.00005 with momentum 0.937.

We selected the YOLOv8 configuration weights from the 3500th epoch for use in our door detector mode. An example output of the YOLOv8 model in the simulation environment is shown in Figure 5.3(a). The corresponding depth perception from the robot's stereo camera, represented as a depth cloud, is illustrated in Figure 5.3(b). While the model performs well overall, the evaluation metrics reveal some false positives, which must be considered when determining door status.



(a) (b)

Figure 5.3. (a) Trained YOLOv8 door detector output, (b) rViz, depth points of robot camera looking through an open door.

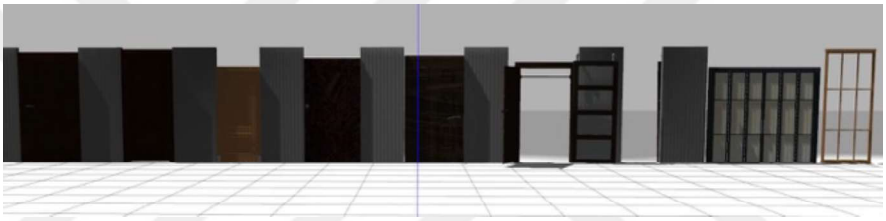


Figure 5.4. Different door mesh models exported to Gazebo.

5.6.2. Door Status Detection Performance

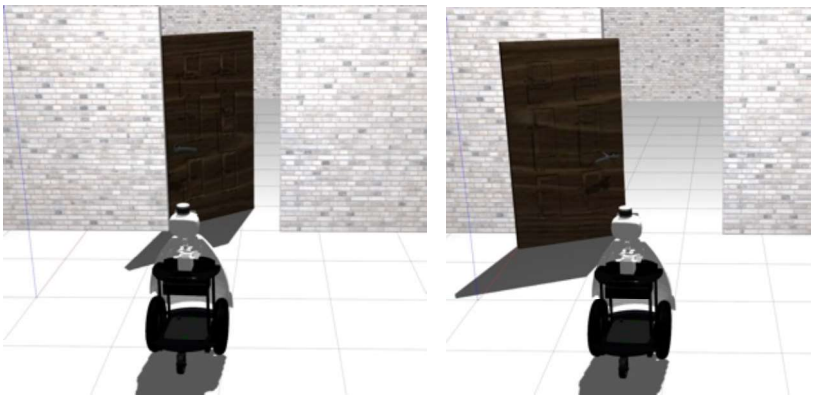
The door-orientation detection algorithm was evaluated in simulation and in real-life data. Due to the dataset's lack of precise ground-truth angles, we performed only qualitative evaluation of its results, while the simulation provides exact door orientation data for qualitative evaluation.

A variety of freely available door meshes as seen in Figure 5.4 are found and imported to the Gazebo environment. For a more accurate simulation of doors, the Gazebo plugin [69] is adapted for Gazebo 11. It can simulate the movement of flip and sliding doors as well as elevators in the case of multi-floor simulation environments. The usage of this plugin is given in Appendix G.5.

Since exact door orientations are available in simulation, we report two scenarios where the robot observes a flip door and computes its orientation: an outward-opening door (Figure 5.5(a)) and an inward-opening door (Figure 5.5(b)). Each scenario is then evaluated using two alternative methods.

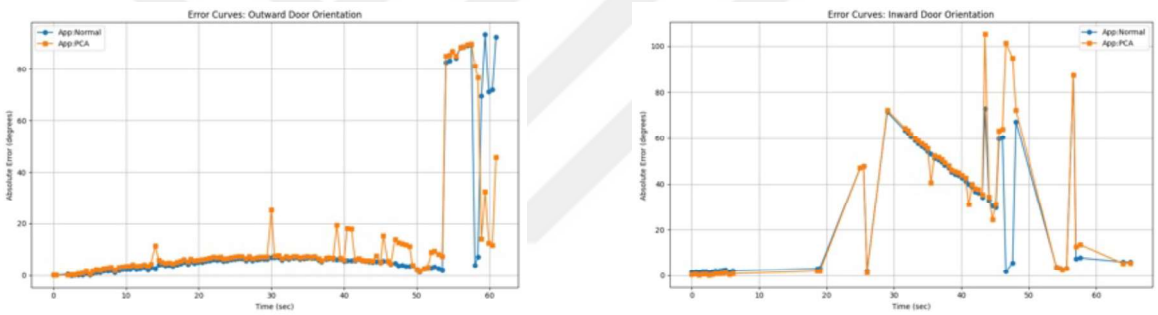
Results for the outward-opening door are presented in Figure 5.6(a) and remain stable until the door's front plane leaves the robot's field of view. In this scenario, the detector performs well, clearly capturing the differences between the PCA method and orientation estimates derived from RANSAC plane normals. The number of detections is indicated by the data points on the plot. Conversely, the detector struggles with inward-opening doors, as reflected by the sparse data points in Figure 5.6(b) and a substantially higher error rate across both methods. From both Figure 5.6 and Table 5.3, the RANSAC plane normal-based approach shows slightly better performance, yielding lower average errors. However, since our pipeline relies heavily on the YOLO detector's output, it remains susceptible to missed door detections caused by unfavorable viewing angles. To address this limitation, constructing a comprehensive door model may be necessary. Incorporating the door's pivot location relative to the robot could enable more accurate orientation estimation, even under challenging perspectives.

Results from the real-life dataset are given for both orientation detection approaches in Table 5.4. Comparative performance of the two orientation detection methods on two different doors are shown in Figure 5.7. While both methods yield similar orientations for one case, as seen in Figure 5.7(a), their outputs are quite different for the latter case, as seen in Figure 5.7(b). This is due to the robot's view of the door that it is looking at; therefore, the approach is also dependent on the view angle and the type of door we looked at.



(a) (b)

Figure 5.5. Door orientation detection simulation environments: (a) outward rotation case, (b) inward rotation case.



(a) (b)

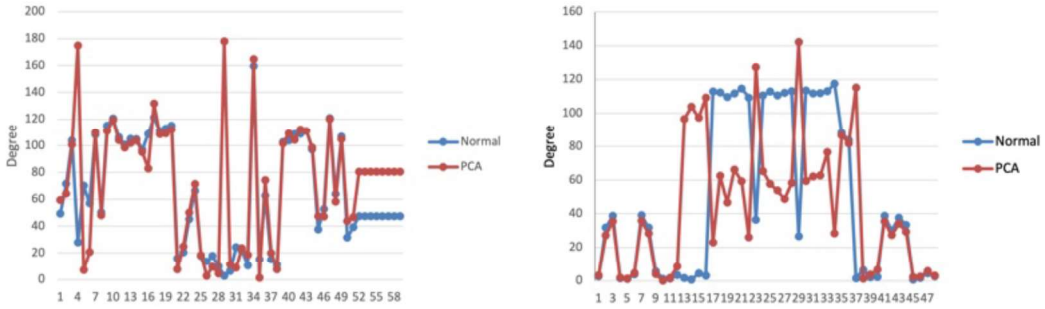
Figure 5.6. Comparative door orientation detection errors in simulation environment: (a) outward rotation case, (b) inward rotation case.

Table 5.3. Door orientation detection: simulation data statistical results.

Rotation	**Method**	**Mean Err. (°)**	**Std. Err. (°)**	**Median Err. (°)**	**Min. Err. (°)**	**Max. Err. (°)**
Outward	DO1	12.8408	24.9749	5.3505	0.0100	93.4126
	DO2	13.4766	22.5703	6.5368	0.1702	89.8548
Inward	DO1	30.4932	25.7460	36.3201	1.2003	87.4075
	DO2	34.1640	29.6230	38.1676	0.1740	105.1640

Table 5.4. Door orientation detection: real-life data statistical results.

	Method	**Mean (°)**	**Std. (°)**	**Median (°)**	**Min. (°)**	**Max. (°)**
Oriented Doors	DO1	30.7730	37.2573	14.4548	0.1996	161.4050
	DO2	59.0061	62.6322	32.5015	0.0000	179.9030



(a) (b)

Figure 5.7. Comparative door orientation detection on door dataset: robot pose (a) looking at outwardly opening door, (b) directly looking through the door.

6. CONCLUSION

This thesis builds upon prior work on the design and development of SempRob, a social mobile robot aimed at enhancing its social capabilities. The significance of this research stems from the growing deployment of service robots in human-populated areas. While navigation in open spaces is relatively straightforward, operating in human-populated environments presents substantial challenges. The robot must respect proxemics - maintaining appropriate social distances - while simultaneously keeping the human target within sight. This thesis addresses this problem through four distinct aspects.

First, an extensive evaluation of the previously proposed Social APF-RL based navigation is conducted in Chapter 2. As explained, Social APF-RL is an extension of the previously proposed APF-RL navigation algorithm while considering proxemics. This method is rigorously tested in both simulation and real-life settings containing non-convex obstacles to validate its effectiveness. In these tests, multiple humans are present to assess the algorithm's social navigation capabilities. While the approach performs well in one-room settings, challenges arise when navigating through multiple-room environments.

Chapter 3 presents the design and development of a social robot head with a pan-tilt mechanism so that the robot can better keep the human target within its field of view. Hardware, control, and software aspects of the design are detailed. The head is also endowed with a face, as explained therein. Finally, the head is then integrated with the robot's body.

Human following is addressed in Chapter 4. First, the social navigation algorithm is revised in several aspects to improve its performance. Next, a human following approach is implemented where the robot's head is actively controlled to maintain visual focus on the target human. Based on the results, a coordinatd control of both

body and head motions is deemed necessary.

Building on the previous Social APF-RL navigation algorithm, a reinforcement learning network has been developed with a reward function designed to incentivize effective following behavior. Although training is ongoing, initial efforts emphasize the following action by applying only minimal penalties for uncoordinated motion. The chapter concludes by presenting the latest model's performance compared to the baseline models with stationary head and with moving head without body-head coordination.

Finally, Chapter 5 considers the problem of recovery in case of a lost human target. It introduces the initiated work on a Kalman filtering-based tracking algorithm, which aims to track the obstructions that may cause harm to our robot's human following task. We base our main obstructions on doors and focus on their detection and status. We base our detection on the YOLO based detection networks, namely, YOLOv8, which was the latest version at the time, and YOLOv4. Started their learning on readily available data. In order to test our trained detector, we collect data from our faculty's doors using a mobile robot. For the determination of the door status, we based ourselves on flip doors and used a pipeline that extracts regions of interest based on the detector output and our RGBD camera. Then, we move on with using a random sample consensus (RANSAC)-based algorithm to extract the planes and the normals of the door and door frame to come up with a door orientation to define its status. We also included a pivot point detection of the door, for which we use the principal component analysis (PCA) to estimate the door normal, which is a vector that corresponds to the smallest eigenvalue of the door cloud. Then, we introduce both simulation and our previously mentioned real-life captured images to see their performance.

For future work, the body-coordination training process must be reconsidered and possibly redesigned to enable a more efficient and effective learning. The trained coordination model must be deployed and evaluated on a real robot to validate its effectiveness in practical scenarios. Up to this point, we have used off-the-shelf trackers;

however comprehensive testing is needed to evaluate how well the tracking system performs in conjunction with the obstruction handling methods, particularly in recovering the lost human target.

Our rresults in door detection and status classification indicate the need for a more accurate door detector. For the door status detection, incorporating prior knowledge about door geometry may enhance the system's ability to interpret viewpoints and their implications for human following behavior. Lastly, beyond doors, other common sources of obstruction - such as corners - should be explicity considered in the target recovery framework to improve robustness in more complex environments.

REFERENCES

1. Bartneck, C. and J. Forlizzi, "A Design-Centred Framework for Social Human-Robot Interaction", *13th IEEE International Workshop on Robot and Human Interactive Communication (IEEE Catalog No.04TH8759)*, pp. 591–594, Kurashiki, Japan, 2004.

2. Mori, M., "Bukimi No Tani [The Uncanny Valley]", *Energy*, Vol. 7, No. 4, pp. 33–35, osaka, Japan, 1970.

3. Mathur, M. B. and D. B. Reichling, "Navigating a Social World with Robot Partners: A Quantitative Cartography of the Uncanny Valley", *Cognition*, Vol. 146, pp. 22–32, 2016.

4. Bektaş, K., *Design of a Social Robot and Safe Social Navigation with Deep Reinforcement Learning*, Master's Thesis, Boğaziçi University, 2022.

5. Bektaş, K. and H. I. Bozma, "APF-RL: Safe Mapless Navigation in Unknown Environments", *2022 International Conference on Robotics and Automation (ICRA)*, pp. 7299–7305, Philadelphia, PA, USA, 2022.

6. Vatan, S. B., K. Bektaş and H. I. Bozma, "Social APF-RL: Safe Mapless Navigation in Unknown Human-Populated Environments", *2023 European Conference on Mobile Robots (ECMR)*, pp. 1–6, Coimbra, Portugal, 2023.

7. Hu, C., X. Ma and X. Dai, "A Robust Person Tracking and Following Approach for Mobile Robot", *2007 International Conference on Mechatronics and Automation*, pp. 3571–3576, Harbin, China, 2007.

8. Algabri, R. and M.-T. Choi, "Deep-Learning-Based Indoor Human Following of Mobile Robot Using Color Feature", *Sensors*, Vol. 20, No. 9, Article 2699, 2020.

9. Toan, N., S.-H. Bach and S. Yi, "A Hierarchical Approach for Updating Targeted Person States in Human-Following Mobile Robots", *Intelligent Service Robotics*, Vol. 16, No. 3, pp. 287–306, 2023.

10. Uğuzlar, U., E. Cansu, E. C. Contarlı and V. Sezer, "Autonomous Human Following Robot Based on Follow the Gap Method", *2023 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pp. 139–144, Tomar, Portugal, 2023.

11. Ren, Q., Q. Zhao, H. Qi and L. Li, "Real-Time Target Tracking System for Person-Following Robot", *2016 35th Chinese Control Conference (CCC)*, pp. 6160–6165, Chengdu, China, 2016.

12. Montesdeoca, J. C., M. Toibero and R. Carelli, "Person-following based on social navigation into the sensorized environments", *2017 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pp. 799–803, Macau, China, 2017.

13. Zhou, W., P. Dickenson, H. Cai and B. Li, "Human Following for Mobile Robots", *Intelligent Robotics and Applications: Lecture Notes in Computer Science*, Vol. 13458, pp. 660–668, Springer International Publishing, Cham, 2022.

14. Caruso, M., E. Regolin, F. J. Camerota Verdù, S. A. Russo, L. Bortolussi and S. Seriani, "Robot Navigation in Crowded Environments: A Reinforcement Learning Approach", *Machines*, Vol. 11, No. 2, Article 268, 2023.

15. Liu, Z., Y. Zhai, J. Li, G. Wang, Y. Miao and H. Wang, "Graph Relational Reinforcement Learning for Mobile Robot Navigation in Large-Scale Crowded Environments", *IEEE Transactions on Intelligent Transportation Systems*, Vol. 24, No. 8, pp. 8776–8787, 2023.

16. Helbing, D. and P. Molnár, "Social Force Model for Pedestrian Dynamics", *Phys. Rev. E*, Vol. 51 (5), pp. 4282–4286, 1995.

17. AWS, "Amazon Web Services (AWS) Robomaker Small House World", , 2019, `https://github.com/aws-robotics/aws-robomaker-small-house-world`, accessed on January 14, 2025.

18. Alfayad, S., M. El Asswad, A. Abdellatif, F. B. Ouezdou, A. Blanchard, N. Beaussé and P. Gaussier, "HYDROïD Humanoid Robot Head with Perception and Emotion Capabilities: Modeling, Design, and Experimental Results", *Frontiers in Robotics and AI*, Vol. 3, Article 15, pp. 1–16, 2016.

19. Hai-bo, Z., Y. Kui and L. Jin-dong, "A Fast and Robust Vision System for Autonomous Mobile Robots", *IEEE International Conference on Robotics, Intelligent Systems and Signal Processing (RISSP 2003)*, Vol. 1, pp. 60–65, Changsha, China, 2003.

20. Ma, J. and J. W. Burdick, "A Probabilistic Framework for Stereo-Vision Based 3D Object Search with 6D Pose Estimation", *2010 IEEE International Conference on Robotics and Automation*, pp. 2036–2042, Anchorage, AK, USA, 2010.

21. Peng, J., A. Srikaew, M. Wilkes, K. Kawamura and A. Peters, "An Active Vision System for Mobile Robots", *Proceedings of the 2000 IEEE International Conference on Systems, Man and Cybernetics (SMC). 'Cybernetics Evolving to Systems, Humans, Organizations, and Their Complex Interactions'*, Vol. 2, pp. 1472–1477, Nashville, TN, USA, 2000.

22. Al-Mutib, K., "Smart Stereovision Based Gaze Control for Navigation in Low-Feature Unknown Indoor Environments", *2014 5th International Conference on Intelligent Systems, Modelling and Simulation*, pp. 121–126, Langkawi, Malaysia, 2014.

23. Tinh, N. V., P. T. Cat, P. M. Tuan and T. T. Q. Bui, "Visual Control of Integrated Mobile Robot-Pan Tilt-Camera System for Tracking a Moving Target", *2014 IEEE International Conference on Robotics and Biomimetics (ROBIO 2014)*, pp. 1566–

1571, Bali, Indonesia, 2014.

24. Chen, H., X. Zhao and M. Tan, "A Novel Pan-Tilt Camera Control Approach for Visual Tracking", *Proceedings of the 11th World Congress on Intelligent Control and Automation*, pp. 2860–2865, Shenyang, China, 2014.

25. Jevtić, A., G. Doisy, Y. Parmet and Y. Edan, "Comparison of Interaction Modalities for Mobile Indoor Robot Guidance: Direct Physical Interaction, Person Following, and Pointing Control", *IEEE Transactions on Human-Machine Systems*, Vol. 45, No. 6, pp. 653–663, 2015.

26. Petrov, P., "An Adaptive Pan-Tilt Camera Control for Visual Target Tracking", *2021 29th National Conference with International Participation (TELECOM)*, pp. 125–128, Sofia, Bulgaria, 2021.

27. Permatasari, D. A. and D. A. Maharani, "Backpropagation Neural Network for Tuning PID Pan-Tilt Face Tracking", *2018 3rd International Conference on Information Technology, Information System and Electrical Engineering (ICITISEE)*, pp. 357–361, Yogyakarta, Indonesia, 2018.

28. Xiao, L., F. Yongchun and Y. Dedong, "Persistent Tracking of a Dynamic Target by a Mobile Robot", *Proceedings of the 33rd Chinese Control Conference*, pp. 2194–2199, Nanjing, China, 2014.

29. Adorni, G., G. Destri, M. Mordonini and F. Zanichelli, "Robot Self-Localization by Means of Vision", *Proceedings of the First Euromicro Workshop on Advanced Mobile Robots (EUROBOT '96)*, pp. 160–165, Kaiserslautern, Germany, 1996.

30. Gockley, R., A. Bruce, J. Forlizzi, M. Michalowski, A. Mundell, S. Rosenthal, B. Sellner, R. Simmons, K. Snipes, A. Schultz and J. Wang, "Designing Robots for Long-Term Social Interaction", *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1338–1343, Edmonton, AB, Canada, 2005.

31. Hu, Y., Z. Zhao, A. Vimal and G. Hoffman, "Soft Skin Texture Modulation for Social Robotics", *2018 IEEE International Conference on Soft Robotics (RoboSoft)*, pp. 182–187, Livorno, Italy, 2018.

32. Grazzani, I., V. Ornaghi, E. Conte, A. Pepe and C. Caprin, "The Relation Between Emotion Understanding and Theory of Mind in Children Aged 3 to 8: The Key Role of Language", *Frontiers in Psychology*, Vol. 9, Article 724, 2018.

33. Mikawa, M., Y. Yoshikawa and M. Fujisawa, "Expression of Intention by Rotational Head Movements for Teleoperated Mobile Robot", *2018 IEEE 15th International Workshop on Advanced Motion Control (AMC)*, pp. 249–254, Tokyo, Japan, 2018.

34. Grossman, G. E., R. J. Leigh, L. A. Abel, D. J. Lanska and S. E. Thurston, "Frequency and Velocity of Rotational Head Perturbations During Locomotion", *Experimental Brain Research*, Vol. 70, No. 3, pp. 470–476, 1988.

35. Ürkmez, M. and H. I. Bozma, "Detecting 3D Hand Pointing Direction from RGB-D Data in Wide-Ranging HRI Scenarios", *Proceedings of the 2022 ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pp. 441–450, IEEE Press, Sapporo, Hokkaido, Japan, 2022.

36. Murtagh, A., "Robot Faces", 2021, `https://github.com/AndrewMurtagh/robot_faces`, a collection of robot faces for robotics applications, accessed on January 15, 2025.

37. Bayoumi, A. and M. Bennewitz, "Efficient Human Following Using Reinforcement Learning", *Proceedings of the Machine Learning in Planning and Control of Robot Motion Workshop (MLCP) at the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1–6, Hamburg, Germany, 2015.

38. Pang, L., Y. Zhang, S. Coleman and H. Cao, "Efficient Hybrid-Supervised Deep Reinforcement Learning for Person Following Robot", *Journal of Intelligent &*

Robotic Systems, Vol. 97, No. 2, pp. 299–312, 2020.

39. Kästner, L., B. Fatloun, Z. Shen, D. Gawrisch and J. Lambrecht, "Human-Following and -guiding in Crowded Environments using Semantic Deep-Reinforcement-Learning for Mobile Service Robots", *2022 International Conference on Robotics and Automation (ICRA)*, pp. 833–839, Philadelphia, PA, USA, 2022.

40. Li, K., Y. Xu, J. Wang and M. Q.-H. Meng, "SARL: Deep Reinforcement Learning Based Human-Aware Navigation for Mobile Robot in Indoor Environments", *2019 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pp. 688–694, IEEE Press, Dali, China, 2019.

41. Kästner, L., T. Buiyan, L. Jiao, T. A. Le, X. Zhao, Z. Shen and J. Lambrecht, "Arena-Rosnav: Towards Deployment of Deep-Reinforcement-Learning-Based Obstacle Avoidance into Conventional Autonomous Navigation Systems", *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6456–6463, IEEE, Prague, Czech Republic, 2021.

42. Kästner, L., C. Marx and J. Lambrecht, "Deep-Reinforcement-Learning-Based Semantic Navigation of Mobile Robots in Dynamic Environments", *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)*, pp. 1110–1115, Hong Kong, China, 2020.

43. Doisy, G., A. Jevtić, E. Lucet and Y. Edan, "Adaptive Person-Following Algorithm Based on Depth Images and Mapping", *Workshop on Robot Motion Planning at the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 43–48, Vilamoura-Algarve, Portugal, 2012.

44. Zhu, Z., H. Ma and W. Zou, "Human Following for Wheeled Robot with Monocular Pan-tilt Camera", *arXiv Preprint arXiv:1909.06087 [cs.RO]*, pp. 1–8, 2019.

45. Khambhaita, H., J. Rios-Martinez and R. Alami, "Head-Body Motion Coordination for Human Aware Robot Navigation", *9th International Workshop on Human-Friendly Robotics (HFR 2016)*, pp. 1–8, Genoa (Genova), Italy, 2016.

46. Rimon, E. D. and D. E. Koditschek, "Exact Robot Navigation Using Artificial Potential Functions", *IEEE Trans. Robotics and Automation*, Vol. 8 (5), pp. 501–518, 1992.

47. Pérez-Higueras, N., "Gazebo SFM Plugin", `https://github.com/robotics-upo/gazebo_sfm_plugin`, 2021, accessed on January 14, 2025.

48. Savarese, F., A. Tejero-de Pablos, S. Quer and T. Harada, "Detecting, Opening and Navigating Through Doors: A Unified Framework for Human Service Robots", *Proceedings of the 14th International Conference on Software Technologies*, ICSOFT 2019, pp. 416–427, SCITEPRESS - Science and Technology Publications, Lda, Setubal, PRT, 2019.

49. Srouji, M., Y.-H. H. Tsai, H. Thomas and J. Zhang, "Human Following in Mobile Platforms with Person Re-Identification", *arXiv Preprint arXiv:2309.12479[cs.RO]*, pp. 1–7, 2023.

50. Hoang, M. D., S.-S. Yun and J.-S. Choi, "The Reliable Recovery Mechanism for Person-Following Robot in Case of Missing Target", *2017 14th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, pp. 800–803, Jeju, South Korea, 2017.

51. Meyer Zu Borgsen, S., M. Schöpfer, L. Ziegler and S. Wachsmuth, "Automated Door Detection with a 3D-Sensor", *2014 Canadian Conference on Computer and Robot Vision*, pp. 276–282, Montreal, QC, Canada, 2014.

52. Shalaby, M. M., M. A.-M. Salem, A. Khamis and F. Melgani, "Geometric Model for Vision-Based Door Detection", *2014 9th International Conference on Computer*

Engineering Systems (ICCES), pp. 41–46, Cairo, Egypt, 2014.

53. Fernández-Caramés, C., V. Moreno, B. Curto, J. F. Rodríguez-Aragón and F. J. Serrano, "A Real-Time Door Detection System for Domestic Robotic Navigation", *Journal of Intelligent & Robotic Systems*, Vol. 76, No. 1, pp. 119–136, 2014.

54. Quintana, B., S. A. Prieto, A. Adán and F. Bosché, "Door Detection in 3D Colored Laser Scans for Autonomous Indoor Navigation", *2016 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, pp. 1–8, Alcala de Henares, Spain, 2016.

55. Prieto, S. A., A. Adán, A. S. Vázquez and B. Quintana, "Passing Through Open/-Closed Doors: A Solution for 3D Scanning Robots", *Sensors*, Vol. 19, No. 21, Article 4740, 2019.

56. Othman, K. M. and A. B. Rad, "A Doorway Detection and Direction (3Ds) System for Social Robots via a Monocular Camera", *Sensors*, Vol. 20, No. 9, Article 2477, 2020.

57. Arduengo, M., C. Torras and L. Sentis, "Robust and Adaptive Door Operation with a Mobile Robot", *Intelligent Service Robotics*, Vol. 14, pp. 409–425, 2021.

58. Ramôa, J., V. Lopes, L. Alexandre and S. Mogo, "Real-Time 2D–3D Door Detection and State Classification on a Low-Power Device", *SN Applied Sciences*, Vol. 3, Article 590, 2021.

59. Antonazzi, M., M. Luperto, N. Basilico and N. A. Borghese, "Enhancing Door-Status Detection for Autonomous Mobile Robots During Environment-Specific Operational Use", *2023 European Conference on Mobile Robots (ECMR)*, pp. 1–8, IEEE, Coimbra, Portugal, 2023.

60. Alparslan, Ö., A. Özcan and Ö. Çetin, "Model ve Veri Odaklı Yaklaşımlar ile Nesne Tespit Başarısının Arttırılmasına Yönelik ESA ve Veri Seti Optimizasyon-

ları", *Mühendislik Bilimleri ve Araştırmaları Dergisi*, Vol. 4, No. 2, pp. 121–128, 2022.

61. Espinoza, N., "Door Detection Dataset", 2022, `https://universe.roboflow.com/nathaly-espinoza/door-detection-zqt59`, accessed on January 14, 2025.

62. Naji, M., "Doors Dataset", 2022, `https://universe.roboflow.com/mohammed-naji/doors-6g8eb`, accessed on January 14, 2025.

63. Zhou, B., H. Zhao, X. Puig, T. Xiao, S. Fidler, A. Barriuso and A. Torralba, "Semantic Understanding of Scenes Through the ADE20K Dataset", *International Journal of Computer Vision*, Vol. 127, No. 3, pp. 302–321, 2019.

64. Mohamed, E. and K. Sirlantzis, "Multi-Environment Object Detection Dataset for Improved Indoor Navigation: Focusing on the Importance of Different Door Handle Types", V1, Mendeley Data, 2023.

65. Kamal, M. O. and B. R. Ahmad, "SRIN: A New Dataset for Social Robot Indoor Navigation", *Global Journal of Engineering Sciences*, Vol. 4, No. 5, Article 596, 2020.

66. Bochkovskiy, A., C.-Y. Wang and H.-Y. M. Liao, "YOLOv4: Optimal Speed and Accuracy of Object Detection", *arXiv Preprint arXiv:2004.10934[cs.CV]*, pp. 1–17, 2020.

67. Varghese, R. and S. M., "YOLOv8: A Novel Object Detection Algorithm with Enhanced Performance and Robustness", *2024 International Conference on Advances in Data Engineering and Intelligent Computing Systems (ADICS)*, pp. 1–6, Chennai, India, 2024.

68. Wang, C.-Y., H.-Y. Mark Liao, Y.-H. Wu, P.-Y. Chen, J.-W. Hsieh and I.-H. Yeh, "CSPNet: A New Backbone that can Enhance Learning Capability of CNN", *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recog-*

nition Workshops, pp. 390–391, Seattle, WA, USA, 2020.

69. Shridhar, M., "Dynamic Gazebo Models", 2014, `https://github.com/MohitShridhar/dynamic_gazebo_models`, accessed on January 14, 2025.

70. Borges, G. and M.-J. Aldon, "Line Extraction in 2D Range Images for Mobile Robotics", *Journal of Intelligent and Robotic Systems*, Vol. 40, pp. 267–297, 2004.

71. Djelal, N., N. Mechat and S. Nadia, "Target Tracking by Visual Servoing", *Eighth International Multi-Conference on Systems, Signals and Devices*, pp. 1–6, Sousse, Tunisia, 2011.

72. Jackermeier, R. and B. Ludwig, "Door Transition Detection for Long-Term Stability in Pedestrian Indoor Positioning", *2019 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, pp. 1–8, Pisa, Italy, 2019.

73. Nonaka, S., S. Nobuhara and K. Nishino, "Dynamic 3D Gaze from Afar: Deep Gaze Estimation from Temporal Eye-Head-Body Coordination", *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2182–2191, New Orleans, LA, USA, 2022.

APPENDIX A: OBSTACLE GENERATION AND MODELLING

The generation of obstacles and their modelling is based on processing the 2D LIDAR data. The robot's LIDAR data is projected to the XY plane to obtain a range sequence $r_{l}{}_{l=1}^{N}$ of laser scan readings with horizontal angular resolution $\Delta\phi$. The method has been revised based on ongoing work in TÜBİTAK TEYDEB 3230750. It consists of three steps:

- First, discontinuity-based clustering is applied.
- Following, an iterative end-point fit algorithm (IEPF) to determine refined clusters.
- Finally, ellipsoids are fit to those clusters using Khachiyan's algorithm.

A.1. Discontinuity-Based Clustering

For each pair of consecutive laser scan points, the adaptive threshold is computed with

$$D_{\max} = \frac{r_{n-1}\sin(\Delta\phi)}{\sin(\lambda_{cluster} - \Delta\phi)} + 3\sigma_r, \tag{A.1}$$

where, $\lambda_{cluster}$ is a predefined angle (e.g. 10°) representing a worst–case incidence and σ_r is the sensor noise standard deviation.

If any consecutive laser scan point exceeds this threshold, a discontinuity is detected. Therefore, the laser scan is segmented where each consecutive point has exceeded this distance threshold.

Algorithm 2 Discontinuity–Based Clustering

Require: Ordered scan points $P = \{p_i = (x_i, y_i)\}_{i=1}^{N}$, ranges $\{r_i\}$, angular step $\Delta\phi$, cluster-angle threshold λ_{cluster}, noise std. σ_r.

Ensure: A set of clusters $S_1, \ldots, S_{N_o}$.

$last_index \leftarrow 1,\ o \leftarrow 1$

for $i = 2, \ldots, N$ **do**

$$D_{\max} \leftarrow \frac{r_{i-1}\,\sin(\Delta\phi)}{\sin\big(\lambda_{\text{cluster}} - \Delta\phi\big)} \;+\; 3\,\sigma_r$$

$$d \leftarrow \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2}$$

if $d > D_{\max}$ **then**

$S_o \leftarrow \{\, p_{last_index}, \ldots, p_{i-1}\}$

$o \leftarrow o + 1, \quad last_index \leftarrow i$

end if

end for

return $\{S_j\}_{j=1}^{o}$

A.2. Iterative End-Pigorithm Fit (IEPF) Algorithm

After discontinuity-based clustering, the initial scan segments are refined using IEPF, also known as the Douglas-Peucker algorithm [70]. The algorithm approximates a line segment to the scan clusters. Given a set of clusters, each cluster has a set of scan points, $P = \{p_1, ..., p_n\}$, where each scan point, $p \in \mathbb{R}^2$. The approximation of the general form of the line segment is given

$$Ax + By + C = 0. \tag{A.2}$$

The terms in the Equation A.2 are defined as, $A = y_M - y_1$, $B = x_1 - x_N$, $C = x_N y_1 - x_1 y_N$. where subscripts indicate the first and last scan points' x and y coordinates. Once the approximated line segment is constructed, we look at the perpendicular distance between the line segment approximation and each scan point with

$$d_i = \frac{|Ax_i + By_i + C|}{\sqrt{A^2 + B^2}}. \tag{A.3}$$

Algorithm 3 Iterative End-Point Fit (IEPF) Algorithm

Require: Point list $S = \{p_i\}_{i=1}^{N}$, tolerance $\epsilon > 0$

Ensure: Simplified polyline approximating S

if $N < 3$ **then**

return S

end if

$(x_1, y_1) \leftarrow p_1, \quad (x_N, y_N) \leftarrow p_N$

$A \leftarrow y_N - y_1, \quad B \leftarrow x_1 - x_N, \quad C \leftarrow x_N\, y_1 - x_1\, y_N$

$d_{\max} \leftarrow 0, \quad \text{idx} \leftarrow 0$

for $i = 2, \ldots, N-1$ **do**

$d_i \leftarrow \left| A\, x_i + B\, y_i + C \right| / \sqrt{A^2 + B^2}$

if $d_i > d_{\max}$ **then**

$d_{\max} \leftarrow d_i, \quad \text{idx} \leftarrow i$

end if

end for

if $d_{\max} \leq \epsilon$ **then**

return $\{p_1, p_N\}$

else

$S_L \leftarrow \{p_1, \ldots, p_{\text{idx}}\}, \quad S_R \leftarrow \{p_{\text{idx}}, \ldots, p_N\}$

return $\text{IEPF}(S_L, \epsilon) \cup \text{IEPF}(S_R, \epsilon)$

end if

The maximum computed distance in a scan cluster is denoted as d_{max}. If d_{max} exceeds the tolerance, ϵ threshold, we break and split the segment at the corresponding index to the d_{max}. Recursively applying this algorithm for each sub-segment, we refine the scan clusters. If the maximum distance does not exceed the threshold, we accept that set of points as a cluster.

A.3. Ellipsoid Fitting (Khachiyan Method)

All obstacles are modeled elliptically. Each ellipse $i = 1, \ldots, N_o$ is defined by five parameters:

- Ellipse center - $\emptyset_i \in \mathbb{R}^2$
- Ellipse major axis length $\mu_i \in \mathbb{R}$
- Ellipse minor axis length $\nu_i \in \mathbb{R}$
- Ellipse orientation $\theta_i \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$

The algorithm essentially considers a polyhedral set and constraints that define the set of those points. Iteratively update the center and the ellipsoid's characteristics to compute the minimum area ellipsoid that encloses the points in the given cluster.

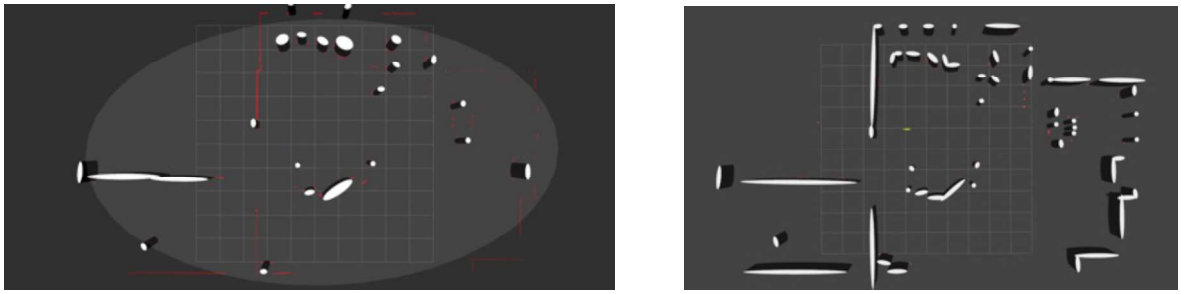
Given N points associated with the cluster at hand, we form the data matrix $D \in \mathbb{R}^{N\times 2}$ using those points. Then the weight vector $u \in \mathbb{R}^N$ is initialized with components $u_i = \frac{1}{N}, i = 1, ..., N$. The vector u will be iteratively updated until it covers all points in the cluster and thus defines the ellipsoid.

The algorithm computes matrix G using the following equation,

$$G = Qdiag(u)Q^T + \lambda_{ellipsoid}I. \tag{A.4}$$

The terms in Equation A.4, $Q = D^T$ and the addition of $\lambda_{ellipsoid}$ term are for the regularization term for handling nearly singular matrices. For each point, the algorithm computes the value g_i to measure the constraint violation that constitutes the given set by the following equation

$$g = Q^T G^{-1} Q. \tag{A.5}$$



(a) Previous method. (b) Revised method, using [70].

Figure A.1. Comparison of obstacle generation methods in AWS Small House [17]: (a) old obstacle generator, (b) revised obstacle generator.

The maximum value of g, given in Equation A.5, is the point where the current fit is not suitable. Based on the found index $i^* = \arg\max\{g\}$ that corresponds to g_i and value $m = \max\{g\}$, it is taken for updating the weights, u. The update on the weights is done with the equation

$$u_{new} \leftarrow (1 - \alpha_{ellipsoid})u, \quad u_{new}(i^*) \leftarrow u_{new}(i^*) + \alpha_{ellipsoid}. \tag{A.6}$$

The terms given in the Equation A.6 are defined as, $\alpha_{ellipsoid} = \dfrac{m-d-1}{(d+1)(m-1)}$ and $d = 2$ since we are looking for a planar ellipsoid. Updating the weights until $||u_{new} - u|| < \epsilon_{ellipsoid}$. The term $\epsilon_{ellipsoid}$ defines the tolerance. Ellipsoid parameters were then extracted using equation

$$o_i = D^T u, \tag{A.7}$$

for the ellipsoid's center, and $o_i \in \mathbb{R}^2$, and for the shape and orientation matrix, E_i. This shape and orientation matrix is obtained by the following equation

$$E_i = \frac{1}{d}(D^T diag(u) D - o_i o_i^T). \tag{A.8}$$

The major and minor axis lengths μ_i, ν_i as well as their orientation θ_i are obtained from the eigenvalues and eigenvectors of the E_i matrix. The pseudo-code of Khachiyan's Ellipsoid Fitting is given in Algorithm 4.

Algorithm 4 Khachiyan's Algorithm for Minimum-Volume Enclosing Ellipsoid

Require: Data $D \in \mathbb{R}^{N\times d}$, tolerance $\epsilon > 0$, regularization $\lambda > 0$

Ensure: Ellipse parameters $(x_c, y_c, b_1, b_2, \theta)$

Initialize $u_i \leftarrow 1/N$ for $i = 1, \ldots, N$

err $\leftarrow \infty$

while err $> \epsilon$ **do**

$Q \leftarrow D^\top$

$G \leftarrow Q\,\mathrm{diag}(u)\,Q^\top + \lambda I$

$g \leftarrow \mathrm{diag}(Q^\top G^{-1} Q)$

$m \leftarrow \max_j g_j, \quad i^* \leftarrow \arg\max_j g_j$

$\alpha \leftarrow \frac{m-d-1}{(d+1)(m-1)} \quad$ (here $d = 2$)

$u_{\text{new}} \leftarrow (1-\alpha)u, \quad u_{\text{new}}(i^*) \leftarrow u_{\text{new}}(i^*) + \alpha$

err $\leftarrow \|u_{\text{new}} - u\|_2, \quad u \leftarrow u_{\text{new}}$

end while

$\mathbf{c} \leftarrow D^\top u$

$E \leftarrow \frac{1}{2}\left(D^\top \mathrm{diag}(u)\, D - \mathbf{c}\,\mathbf{c}^\top\right)^{-1}$

Eigen-decompose E to get eigenvalues b_1^{-2}, b_2^{-2} and eigenvectors

Recover center $(x_c, y_c) = \mathbf{c}$, axes b_1, b_2, orientation θ

A.4. Comparative Obstacle Generation

A comparative case of obstacle generation in the small house environment is presented in Figure A.1(a) and Figure A.1(b) for old and revised obstacle generation results, respectively. As expected, while the previous obstacle generator does not detect most of the walls, the revised method offers a more consistent way of finding the breakpoints and determining the walls as obstacles, along with the doors. This will probably improve the failure cases due to collision given in Table 2.2.

APPENDIX B: SOCIAL APF-RL BASED NAVIGATION

As explained previously, Social APF-Rl based navigation itself is an extension of APF-RL based navigation, which does not account for the humans in the environment [4, 5]. In turn, APF-RL navigation is an extension of APF, which itself is a reactive approach that guarantees safety in a workspace with certain constraints. APF-RL based navigation aims to consider cases in which these constraints do not hold by enabling the respective parameters to be learned using reinforcement learning. Differing from APF-RL, Social APF-RL takes the humans in the robot's surroundings into account and aims to keep its distance from them in accordance with proxemics. The trained model also outputs an expansion parameter that expands the obstacle representations based on the estimated human velocity to prevent the robot's path from intercepting the human path.

B.1. Artificial Potential Functions

Let the robot's center and its goal be denoted as $c, g \in \mathbb{R}^2$ respectively. Furthermore, assume that it has radius $\rho_r > 0$. The potential function $\hat{\varphi} : R^2 \to R$ is defined by equation

$$\hat{\varphi}'(c) \triangleq \frac{\gamma^k(c, g)}{\beta(c, o)}, \tag{B.1}$$

where, $k > 0$. The numerator term $\gamma(c, g)$ is given as

$$\gamma(c, g) \;=\; (g - c)^T (g - c). \tag{B.2}$$

The denominator term $\beta_i : \mathbb{R}^2 \to \mathbb{R}$, in Equation B.1, encodes the obstacles as defined by the set $\mathcal{O}$ with cardinality $n + 1$. The set $\mathcal{O}$ of obstacles is divided into two groups: an external obstacle corresponding to the outer workspace boundary and internal obstacles. Let $n = 0$ denote the outer obstacle and $n \neq 0$ denote the interior obstacles,

the term β, defined as

$$\beta = \prod^{\mathcal{O}} \beta_i^{k_i}(c). \tag{B.3}$$

Robot's distance to each elliptic obstacle as defined by the following equaiton

$$\beta_i(c, o) \;=\; \begin{cases} (\rho_{o_i} - \rho_r)^2 \;-\; \|c - o_i\|^2 & , i = 0, \\ \|c - o_i\|^2 \;-\; (\rho_i + \rho_r)^2 & , i \neq 0 \end{cases}. \tag{B.4}$$

In Equation B.4, the term ρ_i corresponds to the distance between the ellipse center and the boundary point along the line connecting the center of the robot with that of the ellipse. The following equation is the definition of the ρ_i,

$$\rho_i(\theta_i) \;=\; \frac{\mu_i \nu_i}{\sqrt{\mu_i^2 \sin^2(\theta_i) + \nu_i^2 \cos^2(\theta_i)}}, \tag{B.5}$$

where, the θ_i term is the angle that we use to account robot's position, to find the effective radius, ρ_i, is defined as

$$\theta_i(c) = \arctan\left(c_2 - o_{i2}, c_1 - o_{i1}\right)) \;-\; \theta_i. \tag{B.6}$$

Currently, the parameters $k_i = 1$ for every obstacle $i = 0, 1, \ldots, n$. Therefore, the β definition is the same as from previous work. It is formulated so that it can be learned later on.

B.2. Admissibility of APF

This is addressed via changing the definition of φ as follows [46]:

$$\varphi(c) \triangleq \sigma_d \circ \sigma \circ \hat{\varphi}'(c). \tag{B.7}$$

The functions we used from the [46], in the Equation B.7, are $\sigma(x) = \frac{x}{1+x}$ and $\sigma_d(x) = x^{1/k}$. Applying those function combinations results in the following artificial potential function,

$$\varphi(c) \;=\; \frac{\gamma(c,g)}{(\beta+\gamma(c,g)^k)^{\frac{1}{k}}}. \tag{B.8}$$

Theoretically, as the denominator term goes to zero, the $\varphi(c)$ function value will approach one.

B.3. Gradient Computation

The gradient $\nabla_c\varphi(c)$ is computed with

$$\nabla_c\varphi = \frac{(\beta+\gamma^k)^{1/k}\nabla_c\gamma - \gamma\nabla_c(\beta+\gamma^k)^{1/k}}{(\beta+\gamma^k)^{2/k}}. \tag{B.9}$$

The partial derivative terms with respect to c, that contains β, are defined as

$$\nabla_c(\beta+\gamma^k)^{1/k} = \frac{1}{k}(\beta+\gamma^k)^{\frac{1-k}{k}}\nabla_c(\beta+\gamma^k), \tag{B.10}$$

and

$$\nabla_c(\beta+\gamma^k) = \nabla_c\beta + k\gamma^{k-1}\nabla_c\gamma. \tag{B.11}$$

Similarly, the partial derivative term $\nabla_c\gamma(c)$ is equal to

$$\nabla_c\gamma \;=\; -2\,(g-c)\,. \tag{B.12}$$

The partial derivative term, $\nabla_c \beta(c, o)$, in Equation B.11 is given as

$$\nabla_c \beta \;=\; \sum_{i=0}^{n} \left(\left(\prod_{\substack{j=0 \\ j \neq i}}^{n} \beta_j(c,o)^{k_j} \right) k_i \, \beta_i(c,o)^{\,k_i - 1} \, \nabla_c \beta_i(c,o) \right), \tag{B.13}$$

$$\nabla_c \beta_i = \begin{cases} 2\,(\rho_i - \rho_r)\,\nabla_c \rho_i \;-\; \nabla_c \|c - o_i\|^2 & , i = 0, \\ -\,2\,(\rho_i + \rho_r)\,\nabla_c \rho_i \;+\; \nabla_c \|c - o_i\|^2 & , i \neq 0 \end{cases}, \tag{B.14}$$

$$\nabla_c \|c - o_i\|^2 \;=\; 2\,(c - o_i)\,, \tag{B.15}$$

$$\nabla_c \rho_i \;=\; \frac{\partial \rho_i}{\partial \theta_i} \frac{\partial \theta_i}{\partial c}, \tag{B.16}$$

$$\frac{\partial \rho_{o_i}}{\partial \theta_i} = -\,\frac{\mu_i \, \nu_i \, (\mu_i^2 - \nu_i^2)\, \sin(\theta_i)\, \cos(\theta_i)}{\left(\mu_i^2 \, \sin^2(\theta_i) + \nu_i^2 \, \cos^2(\theta_i)\right)^{\frac{3}{2}}}, \tag{B.17}$$

$$\frac{\partial \theta_i}{\partial c} = \begin{pmatrix} -\,\dfrac{c_2 - o_{i2}}{(c_1 - o_{i1})^2 + (c_2 - o_{i2})^2} \\ \dfrac{c_1 - o_{i1}}{(c_1 - o_{i1})^2 + (c_2 - o_{i2})^2} \end{pmatrix}, \tag{B.18}$$

The gradient terms; $\nabla_c \beta_i$, $\nabla_c \|c - o_i\|^2$ and $\nabla_c \rho_i$ in Equation B.13 are given by Equation B.14, Equation B.15, and Equation B.16, respectively. The Equation B.16, contains definitions from Equation B.17 and Equation B.18. The robot's linear and angular velocities are then computed using the direction of $\nabla_c \varphi$. Let $u(t)$ be the velocity

control effort, defined as,

$$u(t) = \begin{bmatrix} v_{\max} \ \cos\left(\angle \nabla_c \varphi(c)\right) \\ \omega_{\max} \ \sin\left(\angle \nabla_c \varphi(c)\right) \end{bmatrix}, \tag{B.19}$$

where,

$$\angle \nabla_c \varphi(c) = \arctan\left(\frac{\partial \varphi}{\partial c_2}, \frac{\partial \varphi}{\partial c_1}\right). \tag{B.20}$$

The $v_{\max}$ and $\omega_{\max}$ terms are the linear and angular speed of the base motors' limits, respectively.

B.4. APF-RL

APF-RL is developed by previous work [4]. The main motivation was to extend the classical APF method's safety to real-world, non-convex environments by having a network that supplies the parameters based on the sensory readings. The network receives the current and previous laser scans and intermediate goals, which it determines. It outputs the k parameter for the classical navigation algorithm APF to manipulate the current topology obtained by the equation B.1, where each k_{β_i} term equals 1. As we mentioned before, it also outputs an intermediate goal term to alleviate the APF's local minima problem, and these intermediate goals are assumed to eventually lead to the final goal.

The DRL model assumes that the underlying environment is structured as a POMDP. It has three hidden layers with 512 neurons each. Below, we list the observations, actions, and reward functions for the model.

- Observations:

$$\omega(t) = \left(q(t),\, q(t-\delta t),\, g(t),\, g(t-\delta t)\right), \tag{B.21}$$

where, $q(t)$ is laser scan at time t, $g(t)$ is intermediate goal at time t.

- Actions:

$$a(t) = \left[k(t),\, g^T(t)\right], \tag{B.22}$$

where, $k(t)$ is gain parameter for attractive field in APF, $g(t)$ is intermediate goal at time t.

- Reward Functions:

$$r\left(a(t), \omega(t)\right) = r_g\left(a(t), \omega(t)\right) + r_o\left(a(t), \omega(t)\right). \tag{B.23}$$

In Equation B.23, the main goal reward function is defined as

$$r_g\left(a(t), \omega(t)\right) = \begin{cases} R_{g1} & \text{if } \delta(t) < \tau_p \\ R_{g2}\dot{\delta}(t) & \text{Otherwise} \end{cases}, \tag{B.24}$$

where τ_p is the distance threshold for reaching the goal, with following definition

$$\delta(t) = |c(t) - g^*(t)|. \tag{B.25}$$

The Equation B.25, defines the robot's proximity to the goal, where $c(t)$ is the position of the robot's base link at time t, and $g(t)$ is the final goal given at time t. The $R_{g_1} = 10$ is the reward we give if our robot reaches the goal, and $R_{g_2} = 2$ is the reward based on the change towards the final goal. Similarly, the obstacle avoidance reward function is defined as

$$r_o\left(a(t), \omega(t)\right) = \begin{cases} R_{o1} & \text{if } \exists\, o \in \mathcal{O}(t) \text{ s.t. } 0 < \beta(c, o) < \tau_o \\ R_{o2} & \text{if } \beta(c, o) = 0 \end{cases}, \tag{B.26}$$

where, τ_o defines a safety zone for obstacles, $R_{o_1} = -0.1$ is the penalty when the

robot reaches an obstacle, and $R_{o_1} = -10$ is when a collision happens between the robot and the obstacle.

B.5. Social APF-RL

Social APF-RL was also developed by previous work [4]; it differs in how it treats humans. It obeys proxemics not to violate the personal space of humans. It also learns a parameter to enlarge the obstacles fitted to humans based on their estimated velocities. Thus, the robot is expected not to intercept the human trajectory.

The DRL model assumes that the underlying environment is structured as a POMDP, similar to APF-RL. It has three hidden layers with 512 neurons each. Below, we list the observations, actions, and reward functions for the model.

- Observations:

$$\omega'(t) = (\omega(t),\, q_h(t))\,. \tag{B.27}$$

- Actions:

$$a'(t) = [a(t),\, \mu_h(t)]\,. \tag{B.28}$$

- Reward Functions:

$$r'(a'(t), \omega'(t)) = r\,(a(t), \omega(t)) + r_h(a'(t), \omega'(t)). \tag{B.29}$$

The new reward, addition to the APF-RL B.4, is the social compliance reward function defined as

$$r_h\,(a'(t), \omega'(t)) = \sum_{i=1}^{N_h} r_{hi}\,(a'(t), \omega'(t))\,. \tag{B.30}$$

The term r_{hi}, in the B.30, is defined as the following equation

$$r_{hi}\left(a'(t),\omega'(t)\right) = \begin{cases} R_h\left(D_h - |c(t) - h_i(t)|\right) & \text{if } |c(t) - h_i(t)| < D_h \\ 0 & \text{Otherwise} \end{cases}. \quad \text{(B.31)}$$

The N_h in the Equation B.31 represents the number of humans, and R_h is set to be -0.6. Similarly, the threshold D_h is the personal zone distance and set to be 1.4 with the ρ_r accounted for. Last term, h_i indicates the ith human position. The elliptic obstacles are expanded based on the new network output μ_h, the following formulas are used to determine new expanded ellipse's parameters

$$\delta b_1 = \mu_h \xi_1 v_h^T e_1, \quad \text{and} \quad \delta b_2 = \mu_h \xi_2 v_h^T e_2. \quad \text{(B.32)}$$

The Equation B.32 terms e_1 and e_2 are the unit vectors of the major and minor axes, respectively. Similarly, the v_h is the estimated human velocity vector, and ξ_i ($i \in \{1, 2\}$), defines the directional preference on the enlargement defined as

$$\xi_1 = \begin{cases} 1 & \text{if } v_h^T e_1 > v_h^T e_2 \\ 0.5 & \text{Otherwise} \end{cases}, \text{ and } \quad \xi_2 = \begin{cases} 1 & \text{if } v_h^T e_1 < v_h^T e_2 \\ 0.5 & \text{Otherwise} \end{cases}. \quad \text{(B.33)}$$

These equations are enlarging the elliptic obstacle without increasing the opposite side of the estimated human velocity. So, their new major and minor axes are defined as:

$$b_1 = b_{1_o} + \delta b_1 \text{ , and } \quad b_2 = b_{2_o} + \delta b_2. \quad \text{(B.34)}$$

where b_{1_o} and b_{2_o} define initial major and minor axis respectively.

APPENDIX C: SOCIAL FORCE MODEL (SFM)

The Social Force Model (SFM) was developed to describe how a pedestrian moves. Similar to APF regarding how it is modeled, it consists of several artificial forces that drive the pedestrian. Those forces, like APF, create a vector field and intend to reflect the individual pedestrian's motivation while traversing the environment.

For ith pedestrian's the velocity v_{ped_i} evolution in time is given by

$$\frac{dv_{ped_i}}{dt} = \underbrace{f_{\text{acc},i}}_{\text{(i) Acceleration Term}} + \underbrace{F_{ped}}_{\text{(ii) Repulsive Forces from Other Pedestrians}} + \underbrace{F_{\mathcal{O}}}_{\text{(ii) Repulsive Forces from Obstacles}} + \underbrace{F_{att}}_{\text{(iv) Attractive Forces}} + \underbrace{\varsigma_i(t)}_{\text{Fluctuation Term}}. \tag{C.1}$$

C.1. Social Force Model (SFM) Force Terms

The five terms given in Equation C.1 is given below:

(i) Acceleration Term:

$$f_{\text{acc},i} = \frac{v^0_{ped_i}\, e_i - v_{ped_i}}{\tau_i}, \tag{C.2}$$

where, v_i^0 is the desired speed of pedestrian i, e_i is the desired direction,v_i is the current velocity, and τ_i is the time it takes to correct deviations from the desired velocity. This term ensures that the pedestrian will accelerate or decelerate to reach the desired velocity.

(ii) Repulsive Forces from Other Pedestrians:
Pedestrians should not collide with each other and should traverse the environment in compliance with the social norms. Therefore, pedestrian i experiences a

repulsive force from every other pedestrian j given by

$$f_{ij} = \omega_{ped} \exp\left(\frac{r_{ij} - d_{ij}}{R_{ped}}\right) \frac{r_i - r_j}{d_{ij}}, \tag{C.3}$$

where, ω_{ped} is a parameter indicates the strength of the repulsion force, and R_{ped} is a range parameter that determines repulsive force's decay to distance, $d_{ij} = \|r_i - r_j\|$ is the distance between pedestrians i and j, $\frac{r_i - r_j}{d_{ij}}$ is the unit vector pointing from pedestrian j to i. The overall repulsive force from all other pedestrians is computed by

$$F_{ped} = \sum_{j \neq i} w_{ij} \, f_{ij}. \tag{C.4}$$

The force is weighted by a directional factor w_{ij} to reflect that pedestrians are more influenced by the force exerted on them from the front side.

(iii) Repulsive Forces from Obstacles:

Not just pedestrians but also the obstacles around the environment exert a repulsive force on the pedestrian i, given by

$$f_{i\mathcal{O}} = \omega_{\mathcal{O}_j} \exp\left(\frac{r_i - d_{i\mathcal{O}_j}}{R_{\mathcal{O}}}\right) \frac{r_i - r_{\mathcal{O}_j}}{d_{i\mathcal{O}_j}}, \tag{C.5}$$

where, $\omega_{\mathcal{O}}$ and $R_{\mathcal{O}}$ are parameters controlling the strength and range decay of the repulsive force due to obstacles, respectively. The $d_{i\mathcal{O}_j}$ is the distance from pedestrian i to obstacle j, $r_{\mathcal{O}_j}$ is the position of the closest point on the obstacle j, r_i is an effective radius for pedestrian i. The overall repulsive force from all other obstacles is computed by

$$F_{\mathcal{O}} = \sum_{\mathcal{O}} w_{i\mathcal{O}} \, f_{i\mathcal{O}}. \tag{C.6}$$

A weight $w_{i\mathcal{O}}$ is applied to the pedestrian-obstacle forces, similar to the pedestrian–pedestrian forces, where the pedestrian gives more importance to those in front of him/her.

(iv) Attractive Forces:

Attractive forces pull pedestrians toward other people, groups, or objects. The attraction between pedestrian i to attraction group or object k is defined by the equation

$$W(d_{ik}) = -\omega_{att} \exp\left(-\frac{d_{ik}}{R_{att}}\right), \tag{C.7}$$

where, $d_{ik} = \|r_i - r_k\|$ is the distance between pedestrian i and the attractive source at position r_k, ω_{att} is a parameter setting the strength of the attraction R_{att} is a parameter determining the range over which the attraction is effective. The attractive force is then obtained as the negative gradient of the potential, $W(d_{ik})$, given by

$$f_{ik}^{\text{att}} = -\nabla_{r_i} W(d_{ik}) = \frac{\omega_{att}}{R_{att}} \exp\left(-\frac{d_{ik}}{R_{att}}\right) \frac{r_k - r_i}{d_{ik}}. \tag{C.8}$$

The unit vector $\frac{r_k - r_i}{d_{ik}}$ directs the force toward the attractive source. The overall attractive forces then are weighted and summed like previous force factors, given as

$$F_{att} = \sum_k w_{ik} \, f_{ik}^{\text{att}}, \tag{C.9}$$

where, directional weight w_{ik} modulates the attractive influence.

(v) Fluctuation Term:

The term $\varsigma_i(t)$ represents random fluctuations or noise in the pedestrian's behavior. This term is added to account for unmodeled noise terms.

APPENDIX D: DOOR DETECTOR EVALUATION

The door detector trained weights' evaluation results are given below as tables and spider charts. The argument table for different configurations of the YOLOv8 is given in Table D.1, where the first row represents the default values for YOLOv8m and - indicates the default value is used for a given configuration. We tested our saved 327 weights with 4646 door images and 1900 negative data that consists mainly of the class of windows, which have similar low features to doors (for detecting the false positives), for a total of 6546 images. We divided the test sets into three: the already available DWSonder [60] dataset, our in-house dataset divided into two, oriented doors, and human following.

For each dataset—Oriented Doors, Human Following, and DWSonder—we list the five best YOLOv4 and YOLOv8 weight files ranked by mAP 50:95, with the numerical results given in paired tables and the same data plotted in paired figures:

- Our Oriented Doors Dataset – Table D.2 and TableD.3; Figure D.1(a) and Figure D.1(b) for YOLOv4 and YOLOv8, respectively for tables and figures.
- Our Human Following Dataset – Table D.4 and Table D.5; Figure D.2(a) and Figure D.2(b) for YOLOv4 and YOLOv8, respectively for tables and figures.
- DWSonder Dataset – Table D.6 and Table D.7; Figure D.3(a) and Figure D.3(b) for YOLOv4 and YOLOv8, respectively for tables and figures.

Note that the YOLOv4 inference is done on CPU while YOLOv8 is done on the GPU; that's why there is a consistent and dramatic time difference between the two models' results on the inference time.

Table D.1. YOLOv8 door detector training arguments for each configuration.

Conf.#	Image Size	Batch Size	Cosine LR Scheduler	LR_0	LR_F	Nominal Batch Size	Optimizer Used
Def. YOLOv8m args.	640	16	FALSE	0,01	0,01	64	auto
Conf. 1	-	-	TRUE	0,02	0,001	128	Adam
Conf. 2	-	-	TRUE	0,001	0,0001	128	Adam
Conf. 3	-	4	TRUE	0,001	0,000075	128	-
Conf. 4	-	-	TRUE	0,005	0,0001	128	Adam
Conf. 5	480	24	TRUE	0,001	0,0001	128	RMSProp
Conf. 6	-	12	TRUE	0,001	0,0001	128	RMSProp
Conf. 7	-	12	TRUE	0,001	0,00001	128	RMSProp
Conf. 8	-	12	TRUE	0,001	0,00005	128	RMSProp
Conf. 9	-	8	TRUE	0,001	0,000075	128	RMSProp
Conf. 10	-	8	TRUE	0,001	0,000075	128	-
Conf. 11	-	32	TRUE	0,001	0,000075	128	AdamW

Table D.2. Top 5 YOLOv4 weights sorted according to mAP50:95 on the oriented doors image set.

Weight Number	mAP50:95	AP50	AP75	AR	F1	FPPI	IoU mean	IoU std	ECE	Avg. Inf. Time
v4_20000	0.49	0.65	0.72	0.02	0.04	1.55	0.79	0.06	0.61	0.55812
v4_100000	0.43	0.52	0.63	0.02	0.04	2.41	0.75	0.11	0.63	0.55553
v4_40000	0.39	0.59	0.60	0.02	0.03	1.11	0.73	0.07	0.68	0.55898
v4_160000	0.38	0.62	0.53	0.05	0.08	5.75	0.72	0.08	0.60	0.55672
v4_150000	0.36	0.40	0.50	0.04	0.06	7.44	0.74	0.09	0.60	0.55895

Table D.3. Top 5 YOLOv8 weights sorted according to mAP50:95 on the oriented doors image set.

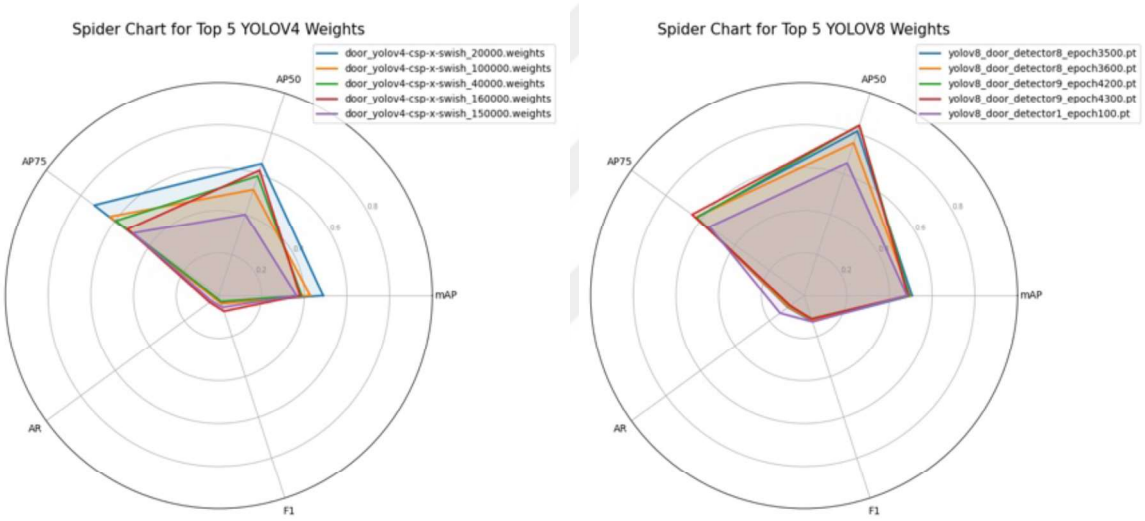
Weight Number	mAP50:95	AP50	AP75	AR	F1	FPPI	IoU mean	IoU std	ECE	Avg. Inf. Time
v8_8_3500	0.51	0.81	0.63	0.09	0.13	6.64	0.77	0.08	0.37	0.00470
v8_8_3600	0.50	0.75	0.63	0.09	0.12	7.25	0.77	0.08	0.37	0.00472
v8_9_4200	0.49	0.84	0.62	0.08	0.11	6.42	0.74	0.07	0.36	0.00246
v8_9_4300	0.49	0.84	0.65	0.08	0.11	5.35	0.75	0.07	0.35	0.00242
v8_1_100	0.48	0.65	0.55	0.14	0.13	17.45	0.79	0.09	0.39	0.00476

Table D.4. Top 5 YOLOv4 weights sorted according to mAP50:95 on the human following image set.

Weight Number	mAP50:95	AP50	AP75	AR	F1	FPPI	IoU mean	IoU std	ECE	Avg. Inf. Time
v4_120000	0.59	0.78	0.77	0.02	0.05	1.72	0.82	0.09	0.57	0.55832
v4_last	0.57	0.66	0.77	0.03	0.05	4.58	0.77	0.11	0.57	0.55924
v4_70000	0.57	0.84	0.76	0.02	0.03	0.69	0.81	0.05	0.53	0.55624
v4_50000	0.55	0.80	0.73	0.02	0.03	1.17	0.80	0.08	0.56	0.55726
v4_130000	0.54	0.83	0.57	0.02	0.04	1.59	0.80	0.09	0.56	0.55442

Table D.5. Top 5 YOLOv8 weights sorted according to mAP50:95 on the human following image set.

Weight Number	mAP50:95	AP50	AP75	AR	F1	FPPI	IoU mean	IoU std	ECE	Avg. Inf. Time
v8_3_300	0.55	0.75	0.66	0.09	0.14	3.67	0.84	0.08	0.58	0.00418
v8_2_800	0.54	0.83	0.58	0.12	0.18	3.98	0.81	0.10	0.53	0.00484
v8_6_1300	0.54	0.72	0.66	0.10	0.17	3.00	0.87	0.07	0.58	0.00484
v8_3_best	0.53	0.74	0.64	0.10	0.15	3.76	0.84	0.08	0.59	0.00424
v8_6_1400	0.53	0.71	0.65	0.10	0.16	3.14	0.87	0.07	0.58	0.00482

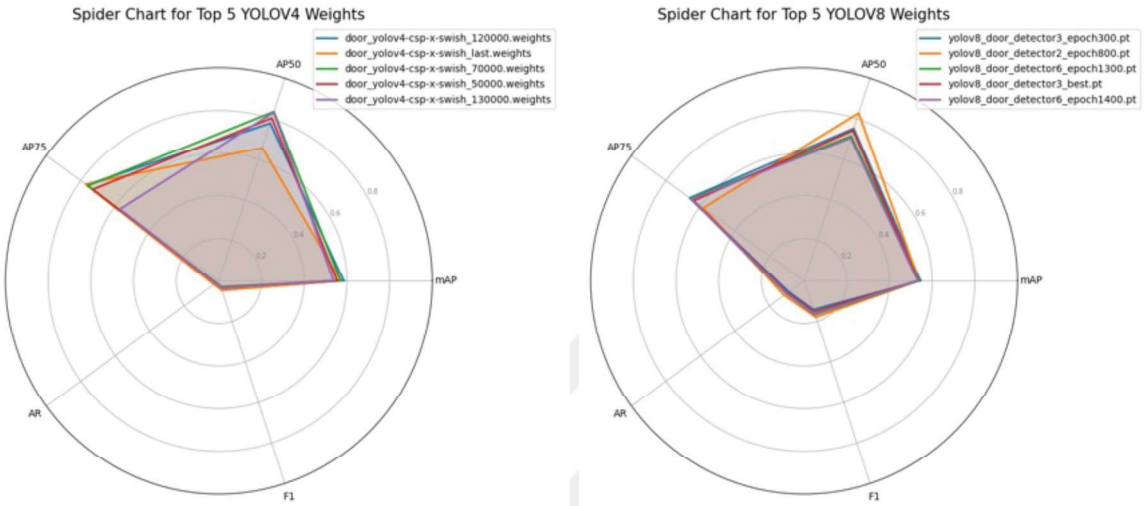


(a) (b)

Figure D.1. Door evaluation spider charts for in-house oriented door test image set: (a) YOLOv4, (b) YOLOv8 results.

Table D.6. Top 5 YOLOv4 weights sorted according to mAP50:95 result on the DWSonder image set.

Weight Number	mAP50:95	AP50	AP75	AR	F1	FPPI	IoU mean	IoU std	ECE	Avg. Inf. Time
v4_120000	0.29	0.41	0.30	0.12	0.12	4.22	0.79	0.09	0.57	0.55808
v4_70000	0.29	0.42	0.31	0.09	0.12	2.16	0.80	0.09	0.59	0.55696
v4_130000	0.26	0.42	0.30	0.11	0.11	3.76	0.79	0.09	0.57	0.55883
v4_140000	0.26	0.40	0.28	0.12	0.12	4.55	0.79	0.09	0.56	0.55856
v4_100000	0.25	0.39	0.29	0.10	0.11	3.68	0.78	0.09	0.56	0.55746



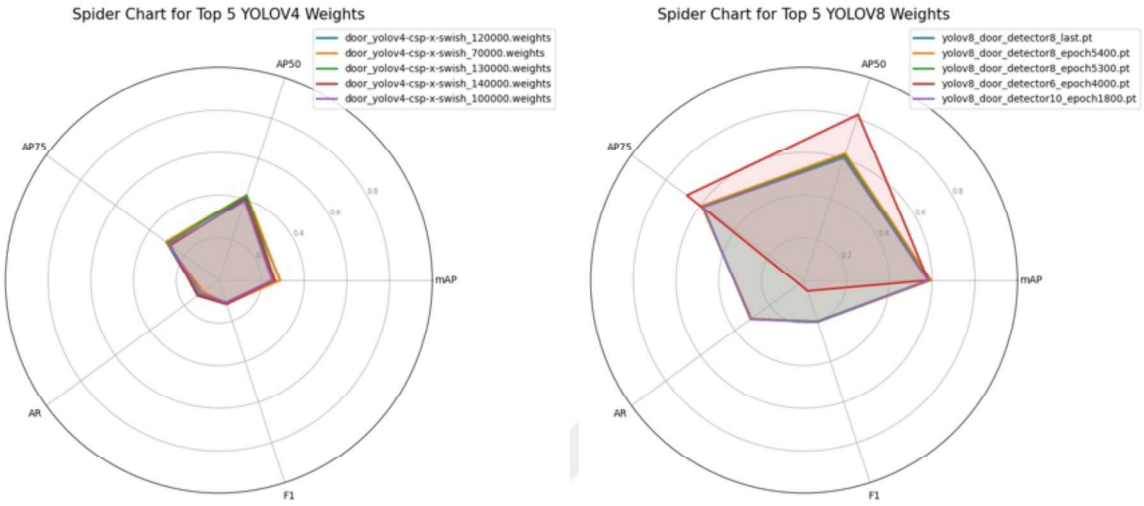
(a) Human following YOLOv4.

(b) Human following YOLOv8.

Figure D.2. Door evaluation spider charts for in-house Human following test image set: (a) YOLOv4, (b) YOLOv8 results.

Table D.7. Top 5 YOLOv8 weights sorted according to mAP50:95 result on the DWSonder image set.

Weight Number	**mAP50:95**	**AP50**	**AP75**	**AR**	**F1**	**FPPI**	**IoU mean**	**IoU std**	**ECE**	**Avg. Inf. Time**
v8_8_last	0.60	0.62	0.59	0.31	0.20	7.87	0.87	0.11	0.37	0.00524
v8_8_5400	0.59	0.63	0.60	0.30	0.20	7.83	0.87	0.11	0.36	0.00522
v8_8_5300	0.58	0.61	0.59	0.31	0.20	8.25	0.87	0.11	0.37	0.00524
v8_6_4000	0.58	0.82	0.68	0.03	0.05	0.29	0.83	0.09	0.65	0.00511
v8_10_1800	0.58	0.60	0.58	0.31	0.20	8.02	0.87	0.11	0.36	0.01133



(a) DWSonder YOLOv4.

(b) DWSonder YOLOv8.

Figure D.3. Door evaluation spider charts for DWSonder test image set: (a) YOLOv4, (b) YOLOv8 results.

APPENDIX E: SIMULATION'S USAGE

This chapter provides an overview of how to use the simulation-related packages of the SempRob. The content is organized into several sections detailing the environment setup, coordination learning, human tracking, social coordination, door detection, free-viewing, and controller topics.

E.1. Environment Setup and Dependencies

E.1.1. System Requirements

- Ubuntu 20.04.6, ROS-Noetic
- CUDA 12.2.91 with CuDNN 8.9.7.29
- OpenCV 4.x built with CUDA and modules

E.1.2. Key Libraries and Tools

- Social Force Model: please refer to https://github.com/robotics-upo/lightsfm .
- SFML for robot_faces API:

  ```
  $ sudo apt install libsfml-dev
  ```

- Darknet: See https://github.com/leggedrobotics/darknet_ros (adjust NVCC flags if necessary; refer to CUDA GPU Compute Capability).
- LibTorch for C++: https://pytorch.org/get-started/locally/ (ensure LIBTORCH_PATH environmental variable is set correctly). The PyTorch version for CUDA 12.2 is not available, but CUDA 12.1 compatible versions work fine with our repository.
- Armadillo: https://arma.sourceforge.net/download.html (for pointcloud fusion and interpolation packages, not explicitly discussed in this thesis).
- Python3: Version 3.8.18 (via conda) with the following packages:

```
$ pip3 install empy==3.3.4 catkin_pkg rospkg numpy
   defusedxml urdf-parser-py
```

Note that for using TensorFlow with GPU enabled, along with CUDA 12.2 DRL environment used Python 3.9.19.

E.1.3. Building the Workspace

The following commands automatically install the remaining, official ROS Noetic packages from the workspace package's dependencies.

```
$ rosdep install --from-paths ${HOME}/SempRob_ws/src --
   ignore-src -r -y
$ sudo apt install ros-noetic-velodyne-* ros-noetic-effort-
   controllers ros-noetic-gazebo_*
$ catkin build
```

E.1.4. Sourcing Setup

The SempRob robot description files must also be sourced for each terminal where we would like to handle robot-related actions.

```
$ source devel/setup.bash
$ source src/semprob_simulation/segway_v3/segway_v3_config/
   std_configs/segway_config_RMP_220.bash
```

E.2. Coordination and APF-RL Learning / Testing

E.2.1. Coord APF-RL Learning Setup

- Dependencies: PyTorch (with CUDA 12.1, compatible with CUDA 12.2), NumPy, Gym 0.26.2, Tensorflow 2.18.0, Stable-Baselines3 2.5.0.
- Training Commands: Below three commands below are to be executed in the

terminal for coordination training. The first command sets up the gazebo and plugins. Later, we initialize the main training code that also spawns the robot by parsing the xacro files, which define the robot description. Lastly, we initialized the controllers and transformations for the robot to start the training process.

```
$ roslaunch apfrl_learning coord_single_human.launch
$ conda activate RL_env
$ roslaunch apfrl_learning main_cpp_segway.launch
$ roslaunch apfrl_learning segway_controllers.launch
```

E.2.2. Using APF-RL Navigation

Any APF-RL model variant can be used using the following lines. The first command's package has various environments; pick one and specify the planar spawn points for the robot. The second line initializes the transform definitions for the robot, then we make a selection based on the desired model, a launch file, from the "apfrl" package.

```
$ roslaunch test_worlds_head_apf case0_indoor_simple.launch
    x_pos:="8.0" y_pos:="0.0"
$ rosrun robot tf2_publisher
$ roslaunch apfrl apfrl_segway.launch
```

E.2.3. Head Tracking

Like APF-RL model testing, we use the first command's package to select a world launch file to set up a Gazebo environment. Case 1 is used because it has a dynamic human in it. Then we use a YOLO-based person detector in the second code, which also sets up the robot transforms. Then we use our head tracking algorithm to track the human using our head mechanism.

```
$ roslaunch test_worlds_head_apf case1_indoor_simple.launch
    x_pos:="8.0" y_pos:="0.0"
```

```
$ roslaunch person_gesture_detector person_detect.launch
$ roslaunch head_apf head_apf.launch
```

E.2.4. Head and APF-RL Simultaneous Usage

The head code can also be used along with our navigation algorithms. For the testing process, we can use pre-determined spawn and goal locations on the rosparam server, as the first commands specify. Then we go on about utilizing the head tracking and navigation algorithms, and then using our goal publisher node to start the process.

```
$ rosrun random_target random_target_node
   _points_to_collect:=5 _goal_file:="src/utility_pkgs/
   targets/goals2.csv" _spawn_file:="src/utility_pkgs/
   targets/spawns2.csv" _test_target_index:=0
$ roslaunch test_worlds_head_apf case2_indoor_simple.launch
    x_pos:=$(rosparam get /random_target_node/x_spawn)
   y_pos:=$(rosparam get /random_target_node/y_spawn)
$ roslaunch person_gesture_detector person_detect.launch
$ roslaunch head_apf head_apf.launch
$ roslaunch apfrl social_apfrl_segway.launch
$ roslaunch goal_publisher coord_simulation_start.launch
```

E.2.5. Human Following with Head Tracking

Like before, we can use a dynamic goal, like a human, with pre-determined spawn locations. This is the same as the testing process, except we pass a human follower to be true to the ROS parameter server. Similarly, any world environment like the corridor world can be used for constructing a simple but relatable corridor-like environment to test how well our model is performing in the human following case.

```
$ rosrun random_target random_target_node
   _points_to_collect:=1 _spawn_file:="src/utility_pkgs/
```

```
   targets/spawns6.csv" _test_target_index:=0 _human_follow
   :=true
$ roslaunch test_worlds_head_apf case6_indoor_simple_follow
   .launch x_pos:=$(rosparam get /random_target_node/
   x_spawn) y_pos:=$(rosparam get /random_target_node/
   y_spawn)
$ roslaunch person_gesture_detector person_detect.launch
$ roslaunch head_apf head_apf.launch
$ roslaunch apfrl social_apfrl_segway.launch
$ roslaunch goal_publisher coord_simulation_start.launch
```

E.3. Door Detection and Segmentation

E.3.1. Door Plugin Test

The door plugin can be tested with the following commands. First, we initialize the example world given in [69]. Then we define the transformations by executing the second command. The last command is for controlling the doors.

```
$ roslaunch dynamic_gazebo_models empty_world_segway.launch
$ rosrun robot tf2_publisher
$ rosrun dynamic_gazebo_models keyboard_op
```

Note: The segmentation code is expected to be integrated into this section.

E.3.2. Door Detector and Status Packages

The trained door packages, along with the imported door meshes, can be tested with the following codes.

```
$ roslaunch detector_depth_door door_env_segway.launch
$ rosrun robot tf2_publisher
```

```
$ roslaunch door_detector_yolov8 door_detect_yolov8.launch
```

The door status and orientation code that uses camera depth information can be used afterward with the following code.

```
$ roslaunch detector_depth_door detector_door_depth.launch
```

E.4. Free-Viewing Code

This part of the code will be utilized in future work to sequentially look at the obstruction. It incorporates a free-viewing motion to the head mechanism to explore the environment, thus it can be used to sequentially eliminate the possible obstructions that cause the robot to lose the human target.

```
$ roslaunch robot attentive_test_vel.launch
$ roslaunch semprob_head_apf semprob_head_apf.launch
```

APPENDIX F: SIMULATION IMPROVEMENTS

Migration from ROS Melodic to ROS Noetic is completed; the differences between Gazebo 9 and Gazebo 11 were adapted throughout this research.

The motor controllers of the pan and tilt units are converted to velocity controllers, with appropriate parameters specified in the MX-64T's datasheet. The pan and tilt unit's description was faulty on the axis definitions, starting from the first link to the camera axis, the proper axis definitions are adopted. Lastly, for PTU, we limit the joint angles, torque, and velocity in the robot description file with the same parameters as the real robot configuration datasheet states. The stereo camera plugin used in the simulation did not reflect the ZED 2 camera we have in real life. The parameters are altered to comply with the ZED2's datasheet and the calibration file that is automatically downloaded by the ZED SDK based on the serial number of the camera we are using. The image size was set to the WVGA option for faster computation, so we also adopted this resolution.

To simulate LIDAR, we adopted the Velodyne plugin and adapted the parameters according to the Robosense RS-LIDAR-16's datasheet. Also, we activated the GPU usage for the LIDAR plugin to measure distance from the non-collision models, like actors that have animation definitions but no explicit collision definitions on the links.

The transform tree was different from the usual convention, normally, one should presume the world to map, map to odom, and odom to the base link of the robot. It is constructed that way, except there was no map, so we directed the transform from world to odom. We are using a local planner approach, mapping is just for the sake of capturing obstructions.

The DRL training was conducted with a turtlebot robot, a subprocess procedure implemented to parse SempRob's xacro files to have the robot description as a ROS

parameter and saved, enabling full control in the simulation regarding spawning and relocating the SempRob model. Thus, making the robot model usage in the training process possible.

APPENDIX G: SIMULATION PLUGINS

G.1. Model Data Plugin

The model data plugin is designed to extract information from the Gazebo simulation to the ROS communication socket. We use Ignition Gazebo libraries to extract the model and its link information in real-time from the simulation. Specifically, static properties (positions, orientations, 3D bounding box enclosures dimensions) and dynamic properties (linear and angular velocities) distinguish dynamic and static models and links.

G.1.1. Plugin Parameters

- topic_name defines the ROS topic on which the model data is published. If not provided in the SDF, the plugin defaults to model_info.
- dt, the elapsed time between simulation iterations, is used to calculate velocities. A small non-zero value is employed to avoid division by zero.
- Model Categorization is done via setting specific names, like Rooms for a room boundary and Room for the internal content of the room. Similarly, actor or dynamic_obstacle are for pre-defining the dynamic models or links. These parts can be altered or extended in the code and can be used as one sees fit (i.e., it can be used for a room complexity calculator code).

G.1.2. Plugin Custom Messages

- Models.msg, is the top-level message, which includes:
  - std_msgs/Header header
  - ModelData[ ] models
- ModelData.msg, contains detailed information about a specific model:
  - string model_name

  - geometry_msgs/Pose model_pose
  - ModelSize model_size
  - uint32 number_of_links
  - LinkData[ ] links
  - LinkData[ ] dynamic_obstacles
- LinkData.msg, contains detailed information about an individual link:
  - string link_name
  - geometry_msgs/Pose link_pose
  - LinkSize link_size
  - geometry_msgs/Twist link_twist
- ModelSize.msg or LinkSize.msg, defines the dimensions of a model or a link, respectively:
  - float64 length
  - float64 width
  - float64 height

G.1.3. Plugin Topics

The model plugin publishes the type Models message, with the topic_name. It does subscribe to Gazebo ROS communication and Gazebo's internal structure with Ignition Gazebo libraries.

G.1.4. Plugin Usage

The workspace source folder requires the plugin package to be included. In your World Spatial Data File (SDF), world file, you should add parameter tags and most crucially specify the topic name with topic_name tag. The detailed information provided by the plugin can be used for:

- Complexity Analysis: Assessing the dynamic behavior and structural complexity of the simulation environment.

- Real-Time Monitoring: Enabling live tracking and debugging of simulation states, which is particularly useful in reinforcement learning applications.

G.2. RL Actor Control Plugin

RL actor control plugin is designed to monitor the actor's state, handle episode resets, and determine valid spawn poses for both the actor and associated robot. The robot's pose is determined based on the target actor, the plugin is specifically designed for human following and we will make sure the robot sees the target at the start. The recovering target is not in the context of this plugin.

This plugin uses a model data plugin to monitor the states of the models in the world to determine suitable spawn points and uses uniform distribution for determining the actor spawn position. Robot spawn is based on proxemics, since we are aiming for the robot to be in the social zone (1.2 to 3.6 m range from the human), the spawn is aimed to be in a public zone divided into two ring zones for spawn point candidates. The first ring is from the 3.6 to 5.6 m range; if there is no available spot, the second ring's range (5.6 to 7.6 m) is sampled. This is for preventing the robot from spawning too far away from the human, but if there are no available spots in these areas (or samples from the uniform distribution happened to not fall into available spots), we use a fallback; the whole room is sampled to find an available pose for the robot.

G.2.1. Plugin Parameters

- topic_name: the name of the topic used to subscribe to the model data plugin's output.
- actor_no: an identifier for the actor, which also links the plugin to a specific room.
- Spawn parameters such as randomize_spawn and respawn_on_episode_reset to indicate if the actor is to be spawned randomly or re-allocated in every episode. respawn_behind_human to have the robot spawn behind the human, if possible. first_waypoint if the respawn_behind_human is set to be true, specify the first

waypoint to indicate the first waypoint of the actor for the computation of the direction of the actor.

G.2.2. Plugin Custom Service Definitions

There are two services, one is a server and the other is the client, given below:

- EpisodeReset.srv defines a server service for resetting an episode for an actor. It includes:
  - Request
    - int64 room_id
    - bool episode_reset
  - Response
    - bool success
    - string message
    - geometry_msgs/Pose robot_pose
- SpawnActor.srv, defines a client service for spawning an actor. It includes:
  - Request
    - geometry_msgs/Pose actor_spawn_pose
  - Response
    - bool success
    - string message

G.2.3. Plugin Services

The plugin advertises a service that handles requests to reset an episode for a given actor. This request (calling the service of EpisodeReset) is given by the reinforcement learning respawn handler script for our case. Service name is /rl_srv/actor$X$/episode_reset, where X corresponds to the actor number provided in the SDF (actor_no). We make these RL plugins modular, so, carrying out actor spawn (via SpawnActor service) and its movements are carried out within the RL actor SFM plugin. Service name is

/rl_srv/actor$X$/spawn_actor, again X corresponds to the actor number. Once we ensure the actor is spawned, based on the response, this plugin carries on determining and sending the suitable robot pose to the RL algorithm.

G.2.4. Plugin Usage

Make sure your SDF implementation is correct and with the RL actor SFM plugin. The services should be able to communicate with each other automatically. RL algorithm should call the appropriate EpisodeReset service.

G.3. RL Actor SFM Plugin

RL actor is based on the plugin from [47], it is entirely responsible for the spawning of the actor models and governing their movement. There are a few contributions to this existing plugin for it to be able to work as intended (usage in RL algorithm). SFM is an old approach for simulating pedestrian movement, so this part can be changed while keeping these improvements to alter pedestrian movement and still be able to be used in an RL algorithm. Now we can communicate the actor's states, like the number of loops, and the episode info related to the actor is also published for the RL network to make use of. Also, respawning the actor during the simulation was made possible by synchronizing the SFM update loop and the plugin itself. One must note that this plugin does not initialize itself if the robot is not in the respective room of the actor.

Lastly, in Gazebo, we have no way of running an identification on the actors due to lightweight graphics in the simulation. We will address the target human as an actor and others as pedestrians (nonhuman-looking models that act the same as the actor). Directly using the SFM plugin [47] in Gazebo for the Model class was not possible, so we wrote another plugin

G.3.1. Plugin Parameters

- actor_no specifies the actor's unique identifier and the room for our implementation.
- velocity, sets the maximum velocity of the actor.
- animation_factor, controls the speed of the walking animation in the Gazebo Simulation.
- people_distance specifies the threshold distance to avoid other pedestrians during the collision avoidance effort.
- Force Weights:
  - goal_weight (sfm.params.forceFactorDesired), weight for the target force.
  - obstacle_weight (sfm.params.forceFactorObstacle), weight for the obstacle repulsion force.
  - social_weight (sfm.params.forceFactorSocial), weight for social forces.
  - group_gaze_weight (sfm.params.forceFactorGroupGaze), weight for group gaze force.
  - group_coh_weight (sfm.params.forceFactorGroupCoherence), weight for group coherence.
  - group_rep_weight (sfm.params.forceFactorGroupRepulsion), weight for group repulsion.
- Group and Obstacles:
  - group, list of model names belonging to the same pedestrian group.
  - ignore_obstacles, list of model names to be ignored during obstacle avoidance. The plugin automatically adds its name and the names of other pedestrians.
- time_step specifies the simulation time step for updates of the social force model.
- Trajectory Parameters, within the trajectory tag below parameters can be specified:
  - cyclic, boolean flag to indicate if the trajectory is cyclic.
  - max_loops, the maximum number of loops to execute before the RL algorithm triggers a reset of episodes.

- episode_timeout, duration after which an episode expires if the actor is non-cyclic. After the last waypoint, the timeout duration is started for the RL algorithm, which triggers a reset of the episode.
- waypoint_threshold, distance threshold to determine when a waypoint has been reached, to count loops.
- waypoint, points to define waypoints on which SFM tries to derive a trajectory.

G.3.2. Plugin Custom Service and Message Definitions

This plugin is using SpawnActor.srv as a server, the definition of it given in the Appendix G.2.2. For messages, we have EpisodeInfo.msg, contains detailed information about an episode:

- int32 episode_no
- int32 current_room
- int32 number_of_loops
- bool non_cyclic_timeout_start
- bool reset_episode_sfm

G.3.3. Plugin Services and Topics

The service name is /rl_srv/actor_control/actor$X$/spawn_actor for serving as spawning actor. Based on this, it responds RL actor plugin to determine a suitable robot spawn pose. The plugin advertises on /rl_msg/actor$X$/episode_info (with X representing the actor ID) to communicate the actor's status, loop count, and reset signals back to the RL algorithm. Similarly, it receives /rl_msg/rl/episode_info to receive episode state updates (e.g., current room and loop count) from the RL algorithm. This communication allows us to reason about when to end or evaluate the simulation episode, and its message is EpisodeInfo.msg.

G.3.4. Plugin Usage

The plugin should work as intended once it is appropriately loaded. The actor control plugin and RL SFM should be able to work with each other. RL algorithm must subscribe to the actor's episode info topic and advertise its episode info.

G.4. Move Model Plugin

The plugin either simply moves a model between the pre-defined waypoints with a given velocity or utilizes the SFM and acts like a non-human-looking pedestrian.

G.4.1. Plugin Parameters

- Common parameters:
  - velocity, desired velocity of the model
  - trajectory, a list of waypoint elements defining the model's path.
- SFM Parameters:
  - use_sfm, when true one must specify same parameters in Appendix G.3 under this tag.

G.4.2. Plugin Usage

The plugin will simply move the Gazebo Model between the pre-defined waypoints. Common parameters like velocity and waypoint tags in trajectory must be set. This plugin will also check if the robot is in the room or not.

G.5. Dynamic Gazebo Models Plugin

This plugin is available in [69], and it is modernized for use in Gazebo 11. It is designed to dynamically control the door and elevator models by establishing ROS-Gazebo communication.

G.5.1. Plugin Parameters

- door_type: Specifies the door type (e.g., flip or sliding).
- door_direction: Defines the rotation direction, CW or CCW.
- model_domain_space: A naming prefix used for the door model that the plugin initializes to.

G.5.2. Example Model Definition

Below is an SDF snippet for a flip door model using hinge links and a revolute joint. Notice how the door has two links (the hinge and the door) and a revolute joint door_hinge that connects them. The plugin is included as follows:

Listing G.1. Abstract SDF snippet for a Door Model with essential plugin and hinge elements

```
<model name="door_1">
  <!-- Hinge link  -->
  <link name="hinge">
    <pose>${-door_w/2}  0 0 0 0 0</pose>
    ...
  </link>

  <!-- Door link -->
  <link name="door">
    <pose>${door_w/2} 0 0 0 0 0</pose>
    ...
  </link>

  <!-- Revolute joint connecting hinge and door links -->
  <joint name="door_hinge" type="revolute">
    <parent>hinge</parent>
    <child>door</child>
```

```
    <pose>0 0 0 0 0 0</pose>
    <axis>
      <xyz>0 0 1</xyz>
      ...
    </axis>
  </joint>

  <!-- Overall model pose -->
  <pose>${pos_x} ${pos_y} ${pos_z} 0 0 ${yaw_door/2}</pose>

  <!-- Plugin to control door dynamics -->
  <plugin name="door1_plugin" filename="libdoor_plugin.so">
    <door_type>flip</door_type>
    <door_direction>clockwise</door_direction>
    <model_domain_space>door_</model_domain_space>
  </plugin>
</model>
```

G.5.3. Dynamics Manager

The plugin works with the dynamics manager node that it contains, it provides an interface between ROS and the Gazebo model via interacting with the plugin. One can add the following launch file content to their own launch file for initializing management on the plugin.

Listing G.2. Launch file snippet for dynamic gazebo models

```
<launch>
    ...
  <!-- Model Dynamic Manager -->
  <arg name="add_group_srv" value="/model_dynamics_manager/
     add_control_group"/>
```

```
  <node pkg="dynamic_gazebo_models" type="dynamics_manager"
      name="dynamics_manager" output="screen"/>
    ...
  <!-- Add Control Group for Flip Doors -->
  <node pkg="rosservice" type="rosservice" name="
     flip_door_group" args="call
    $(arg add_group_srv)
    '{
      group:
      {
        group_name: 'FlipDoors',
        type: 'door',
        active_units: [1, 2, 3, 4]
      }
    }'
  "/>
    ...
</launch>
```

As one can see, the control group is initialized for the flip door; for sliding door or elevator, you can refer to the [69].