Projects Presentation

Jose Peeterson

Internship: RL-collision avoidance

Overview

- Robot's travel times to reach their goals depend on a list of possible trajectories from start to goal position, robot's velocities at each step & collision radius to other robots and obstacle.
- To find the optimal policy shared by all robots, Policy gradient based RL called proximal policy optimisation is used to minimize the expectation of the mean travel time of all the robots.

$$\underset{\pi_\theta}{\operatorname{argmin}} \ \mathbb{E}\left[\frac{1}{N}\sum_{i=1}^{N} t_i^g \middle| \pi_\theta\right],$$

- Using the observation each robot independently computes an action sampled from the shared policy.

3) Reward design: Our objective is to avoid collisions during navigation and minimize the mean arrival time of all robots. A reward function is designed to guide a team of robots to achieve this objective:

$$r_i^t = ({}^g r)_i^t + ({}^c r)_i^t + ({}^w r)_i^t. \quad (4)$$

The reward r received by robot i at timestep t is a sum of three terms, ${}^g r$, ${}^c r$, and ${}^w r$. In particular, the robot is awarded by $({}^g r)_i^t$ for reaching its goal:

$$({}^g r)_i^t = \begin{cases} r_{arrival} & \text{if } \|\mathbf{p}_i^t - \mathbf{g}_i\| < 0. \\ \omega_g(\|\mathbf{p}_i^{t-1} - \mathbf{g}_i\| - \|\mathbf{p}_i^t - \mathbf{g}_i\|) & \text{otherwise.} \end{cases} \quad (5)$$

When the robot collides with other robots or obstacles in the environment, it is penalized by $({}^c r)_i^t$:

$$({}^c r)_i^t = \begin{cases} r_{collision} & \text{if } \|\mathbf{p}_i^t - \mathbf{p}_j^t\| < 2R \\ & \text{or } \|\mathbf{p}_i^t - \mathbf{B}_k\| < R \\ 0 & \text{otherwise.} \end{cases} \quad (6)$$

To encourage the robot to move smoothly, a small penalty $({}^w r)_i^t$ is introduced to punish the large rotational velocities:

$$({}^w r)_i^t = \omega_w |w_i^t| \quad \text{if } |w_i^t| > 0.7. \quad (7)$$

We set $r_{arrival} = 15$, $\omega_g = 2.5$, $r_{collision} = -15$ and $\omega_w = -0.1$ in the training procedure.

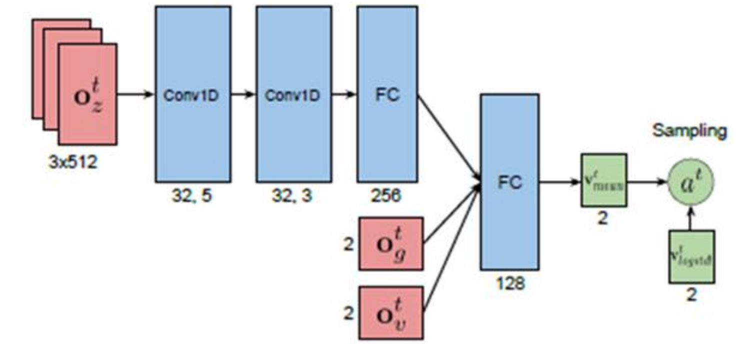
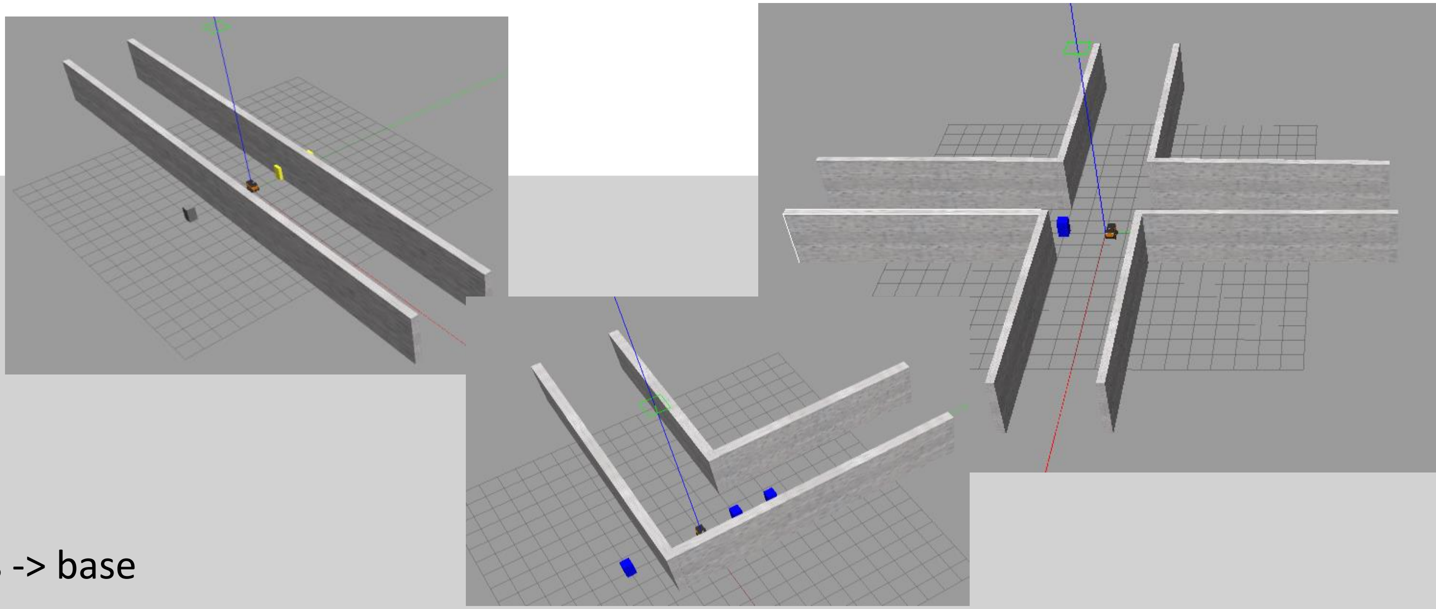


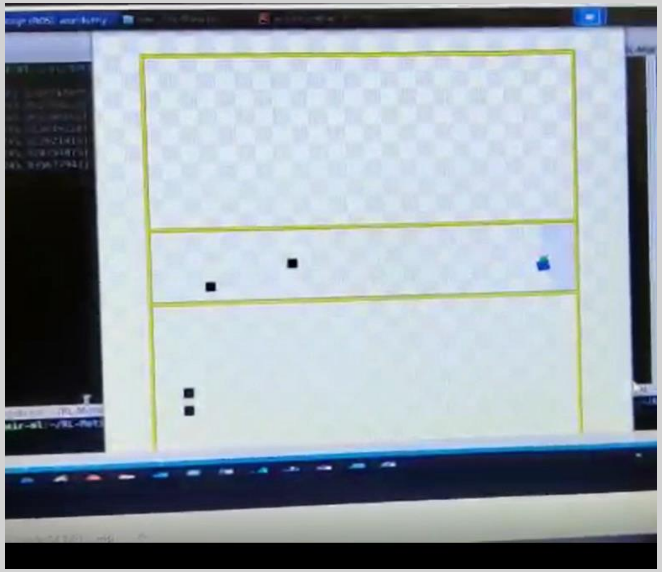
Fig. 3: The architecture of the collision avoidance neural network. The network has the scan measurements $\mathbf{o}_z^t$, relative goal position $\mathbf{o}_g^t$ and current velocity $\mathbf{o}_v^t$ as inputs, and outputs the mean of velocity $\mathbf{v}_{mean}^t$. The final action $\mathbf{a}^t$ is sampled from the Gaussian distribution constructed by $\mathbf{v}_{mean}^t$ with a separated log standard deviation vector $\mathbf{v}_{logstd}^t$.

Closed Environment simulation setup for testing



- Created .launch and .world files for Gazebo and Stage.
- 20 scenes with static and dynamic obstacles in perpendicular, towards and adjacent directions.
- **env** -> base
  - Code publishes initial position message to robot over topic /cmd_pose
  - Code publishes linear and angular velocity message to robot over topic /cmd_vel
  - Code subscribes to laser scans, robot's odometry, crash topic.
  - deepcopy last 3 laser scan readings.
  - Generate goal point and place visualization marker
  - Compute reward
- **env** -> base -> scene_all
  - Check if robot crashed
  - Data logging such as robot pose at every step, distance and steps per episode
  - Decide robot starting point as per world and reset robot (initial) pose
  - Decide goal point as per world
  - Set obstacle class according to world

- **obs** -> base
- Code publishes initial position message to obstacle over topic /cmd_pose
- Code publishes constant linear velocity message to obstacle over topic /cmd_vel
- Code subscribes to obstacles odometry and crash topic.
- clear obstacles at the end of a episode
- Setup static and dynamic obstacles both with some uniform random offset according to world
- check if obstacles crashed
- **obs** -> base -> corridor_straight , corridor_l
- Set obstacles starting point and velocity according to world
- Check wall crash to enable bouncing off walls for obstacles



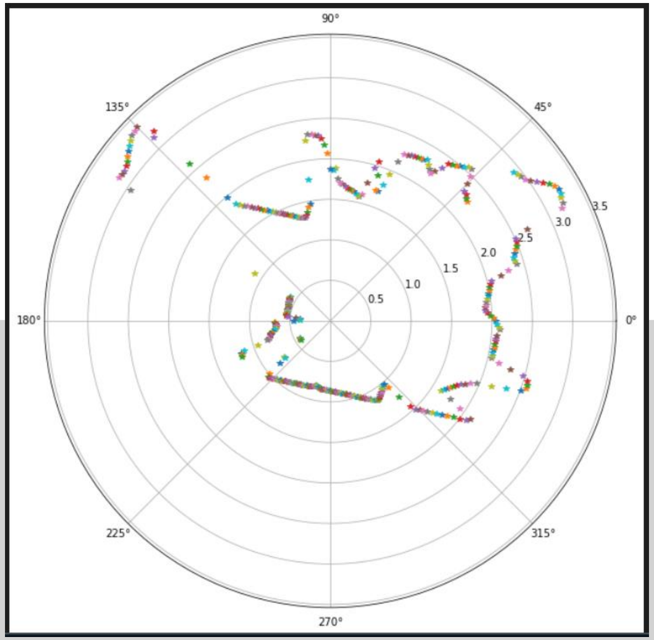
Simulation Observation and results

- Undertraining issues – Robot easily crashed when navigating through obstacles when reaching goal
- Overtraining issues – Robot had a wriggling motion, so it got stuck in local minima near the goal at 4m when using policy trained for 1000 episodes.
- Robot developed a Left bias due to non-random initialization of robot and positioning goal on left side.

Simulator = stage	**Policy**	**Total**	**Success**	**Time out**	**collision**
Scene_straight 20200921-124053	corridor_narrow_00500	**1100** (3300)	100+100+100+100+92+72+88+75+72+57+78= **934** (2810) 85.2%	0	8+28+12+25+28+43+22 =**166**(490)
Scene_l 20200921-124536	corridor_narrow_00500	4200	3100 74%	26	1074
Scene_plus 20200921-125135	corridor_narrow_00500	5100	4238 83%	69	793

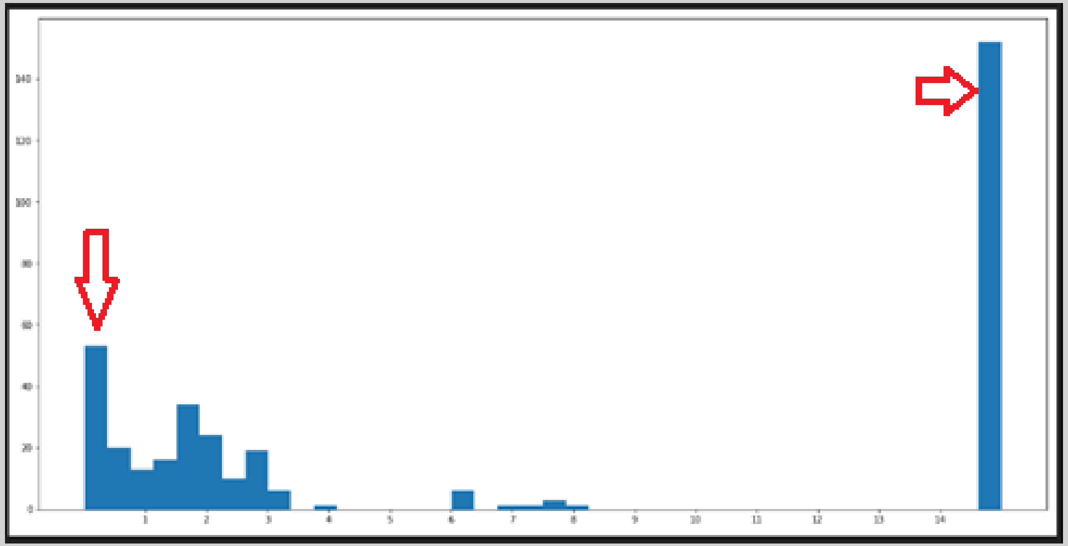
Simulator = Gazebo	**Policy**	**Total**	**Success**	**Timeout**	**collision**
Scene_straight 20200921-130728	corridor_narrow_00500	3300	2833 85.8%	4	463
Scene_l 20200922-110813	corridor_narrow_00500	4200	3058 73%	51	1091
Scene_plus 20200922-111746	corridor_narrow_00500	5100	4198 82.3%	90	812

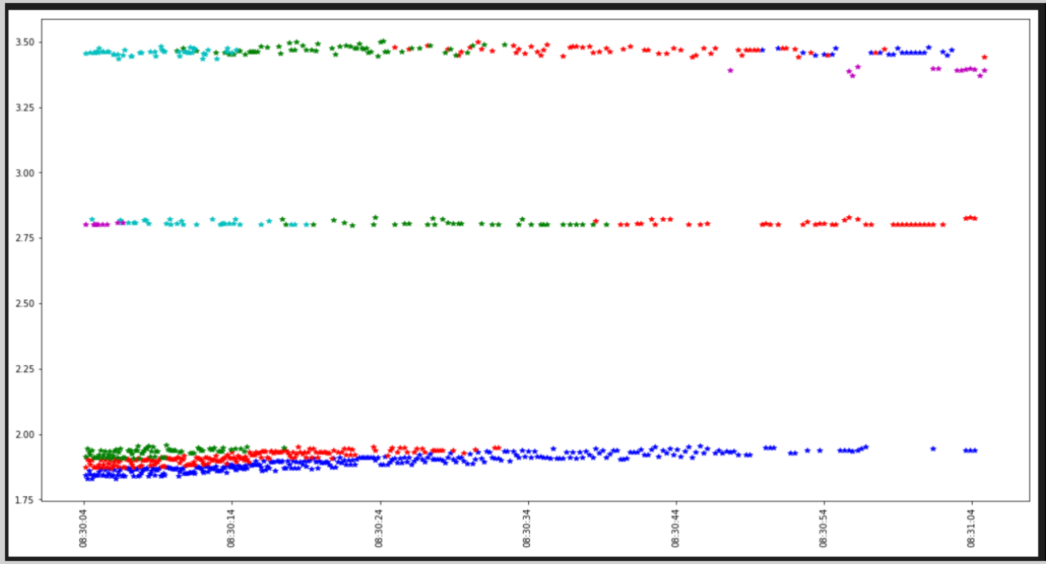
2D HW Lidar



- Lidar calibration in a controlled environment is a covered box of known dimensions and without transparent and non-glass surfaces.
- **Objective:** 1) Model gaussian noise in a beam for injection during model training.
  2) Detect any dead zones.
- **Observation** :
  - Dead zones in scans for motor_pwm below setting of '320'.
  - On the same beam - Jittering noise and outlier noise.
  - Beam overlap - For scene boundaries, readings jump from one beam to another
- **Result:** motor_pwm speed was '530' as it creates 10Hz motor speed/scan rate.

(1.9952305412578 01, 0.3902603748115385, 'angle = ', 130)	Blue
(2.3525734281206465, 0.6371830809427076, 'angle = ', 131)	Red
(2.614447675472082, 0.6860679163401747, 'angle = ', 132)	Green
(3.205831171630265, 0.3173995150590043, 'angle = ', 133)	Cyan
(3.150363662026145, 0.2891222457999423, 'angle = ', 134)	magenta





2D HW Lidar

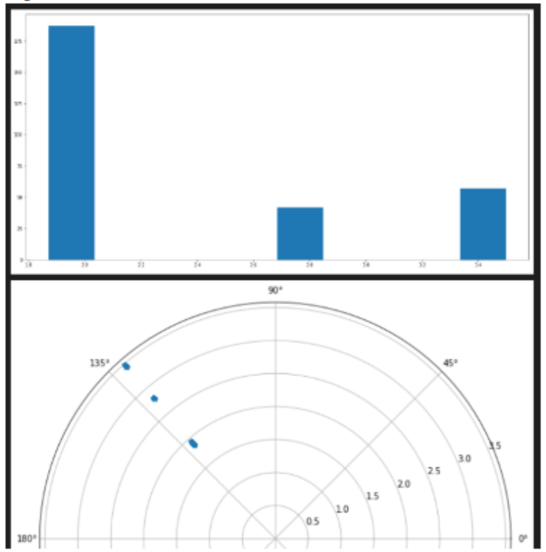
Goal: Show that STD after doing K-means in open case is comparable to the box case

Perform K-means for 2 and 3 clusters

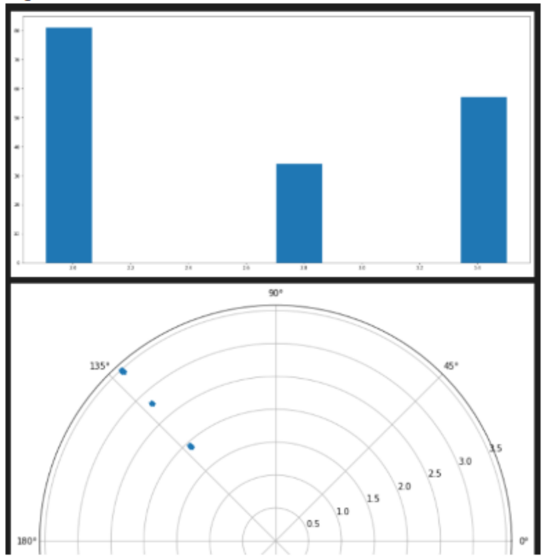
	STD box	STD open	Mean (2-cluster)	STD(2-cluster)	Mean (3-cluster)	STD(3-cluster)
Angle where STD > 0.6						
131	0.00227304497	0.63718308809427076	1.91065	0.021185473474898683	1.91065	0.021185473474898683
132	0.00154578666	0.6860679163401747	3.2227	0.32200581908647735	1.93102	0.013839446708734828
147	0.00807804848	0.71362086453655594	3.1695	0.04682519970770189		('angle = ', 147, 'True clusters = ', 2)
148	0.00111822833	1.04414445724378373	0.5883	0.0030510600689692526		('angle = ', 148, 'True clusters = ', 2)

- K-means clustering – cluster centres are initialized from 1 to 5 clusters. Minimum Change in total intra-cluster variation is used to detect true number of clusters after initializing with 1 to 5 cluster centres.
- Cluster with maximum number of points selected as true cluster and its standard deviation was used in jittering noise estimation.

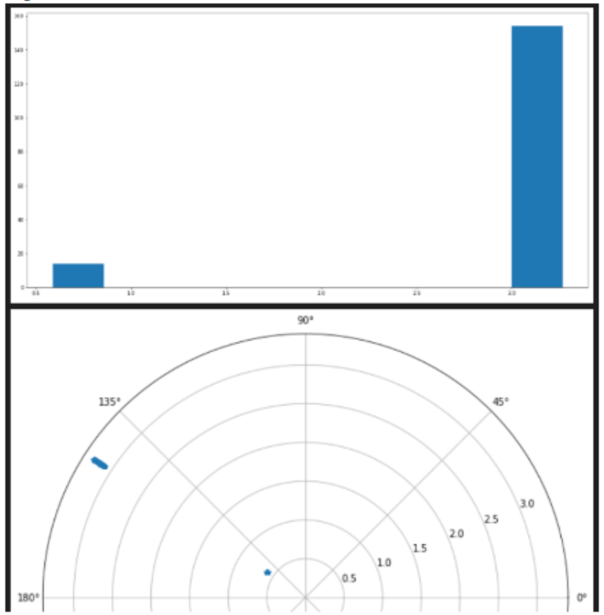
Angle = 131



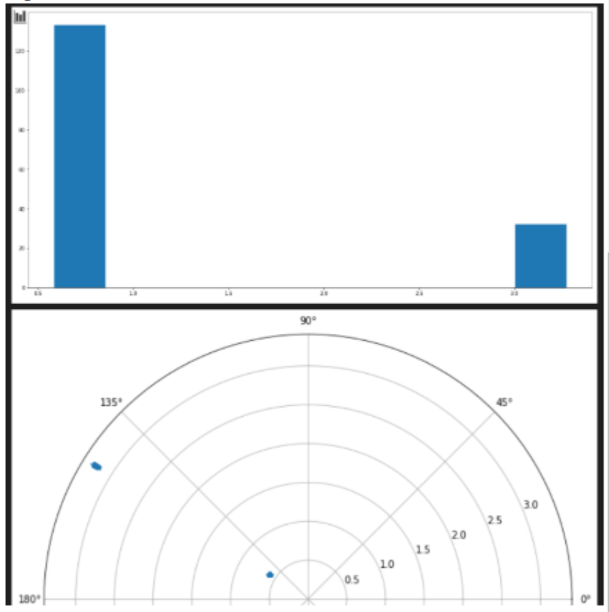
Angle = 132



Angle 147



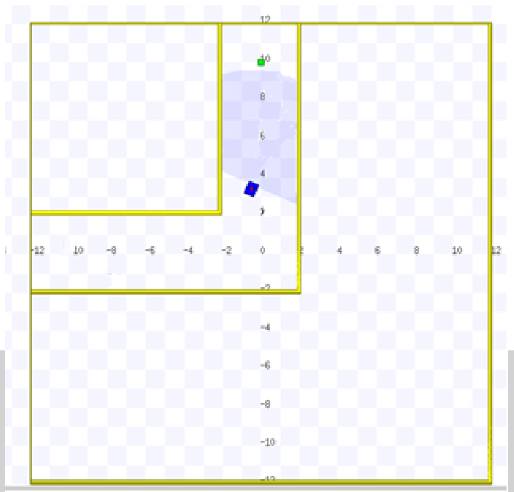
Angle 148

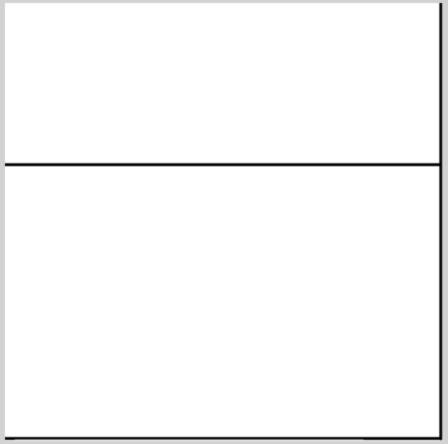


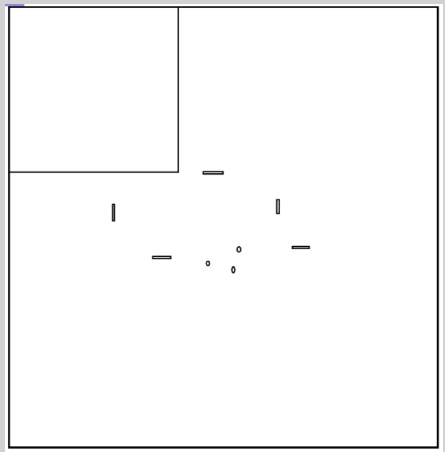
Indoor-Outdoor classifier

- The robot's environment information maybe used for collision avoidance by limiting the linear and angular velocities.
- Dataset creation: 6 maps, 4 obstacle types, 9 or 10 positions , 10 rotations

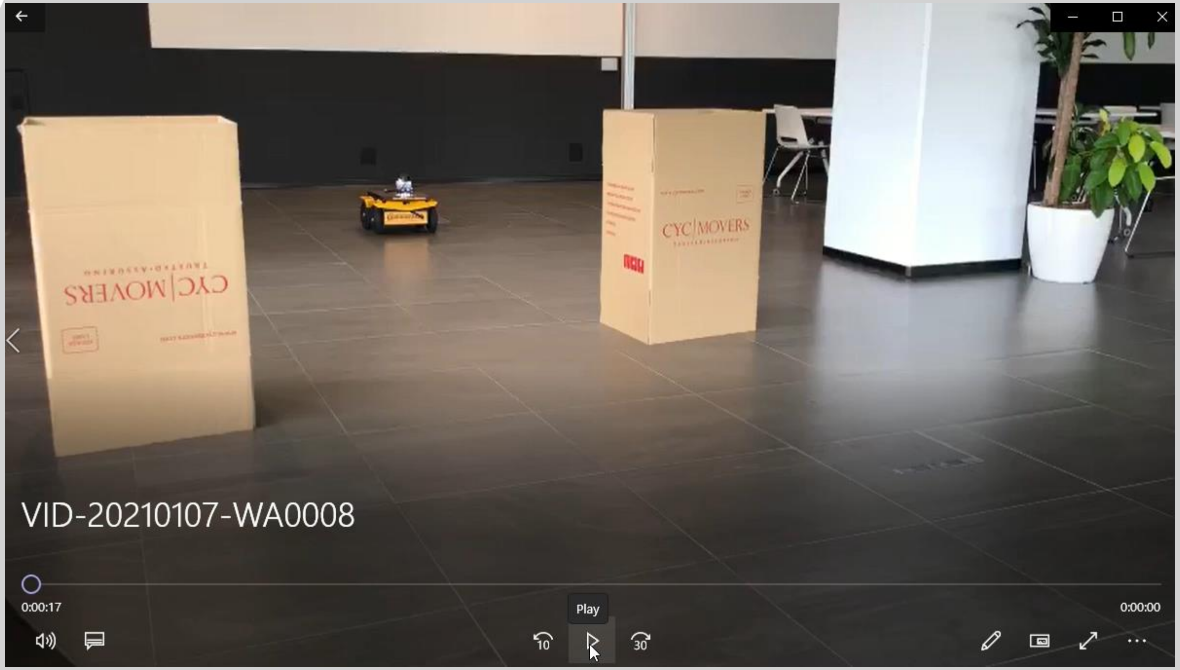
	Logistic regression	SVM
50%-50% train-test split along robot position	98%	95%
Train with high density obstacle and test with low density obstacle type	98%	98%
Train with high density obstacle and test with low density obstacle type	98%	91%
Use different combination of Indoor-outdoor maps for training and testing.	98%	97%

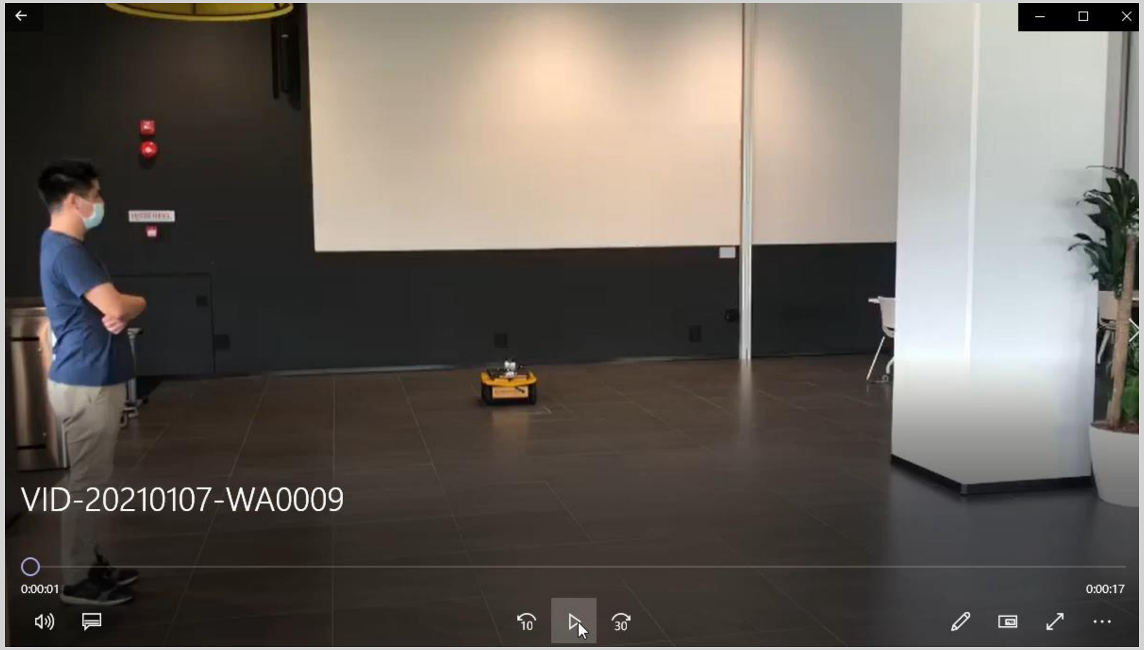






Demo in JACKAL Robot





The END

Testing pseudocode outline

Main.py

Initialize the logger -> folder name, file name logging format, config file

Create object of class Agent

agent.run()

Agent.py

```
for each scene (1-20)
  load_environment(scene_config_file)
      Create object of env class stageworld()
      Initialize logging for this scene

run_batch(policy)

  get_state_observations() # At start of episode get last 3 latest observations, position of goal w.r.t. robot in polar coords. velocity of robot.

  While not terminal_flag and not rospy.is_shutdown(): # LOOP till end of episode. (success, timeout or crash)
     state_list = comm.gather(state, root=0) # gather state observations from all robots
     policy.generate_action(state_list) # each robot independently computes an action from a shared policy
     comm.scatter(scaled_action, root=0) # scatter actions to respective robots

     step+=1 # count the step

     reward, terminal_flag, result = get_reward_and_terminate(steps, crash_mutual) # for every step, check end of episode

     ep_reward += reward        # count total reward in this episode

     r_list = comm.gather(r, root=0)
     terminal_list = comm.gather(terminal, root=0) # gather rewards and terminal flag from all the robots

     get_state_observations() # within the loop for every step.

  self.logger.info = ('  ')  # after episode ends, log info. about episode_id, no. of steps, reward and result.
```