# Safe Control and Reinforcement Learning for Autonomous Navigation

Keyron Linarez, Nicolas Hernandez, Andrew Clark, Hongchao Zhang
Electrical and Systems Engineering Department, McKelvey School of Engineering, Washington University in St. Louis, St. Louis, MO 63130
Cyber-physical System Trustworthiness Lab

1

## Problem and Motivation

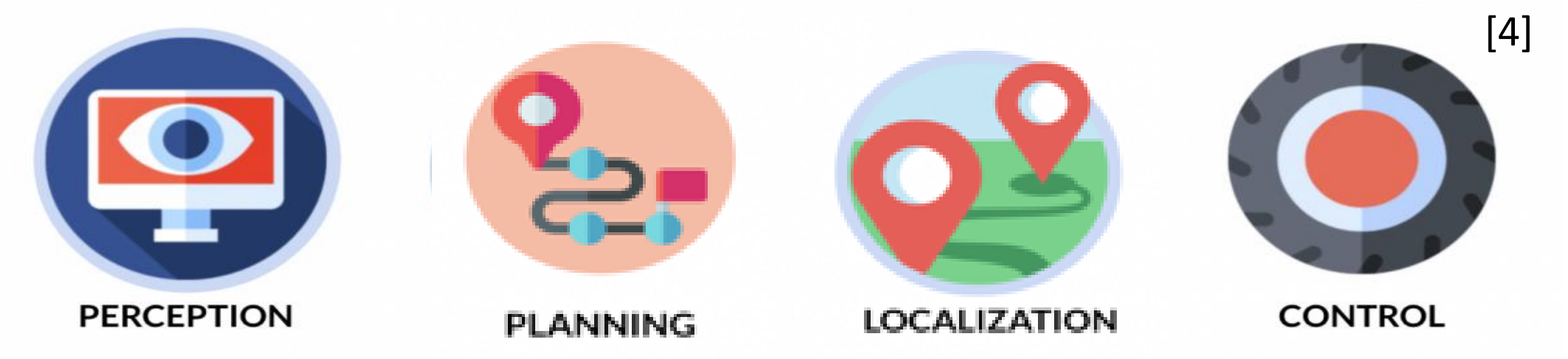
- **Objective:** Develop a navigation system that safely and efficiently avoids obstacles in a maze
- **Constraint:** Avoid collisions with wall sand obstacles
- Research efficient control and path planning methods
- Utilize reinforcement learning to train an agent to minimize proximity to walls and obstacles.

2

## Materials and Methods

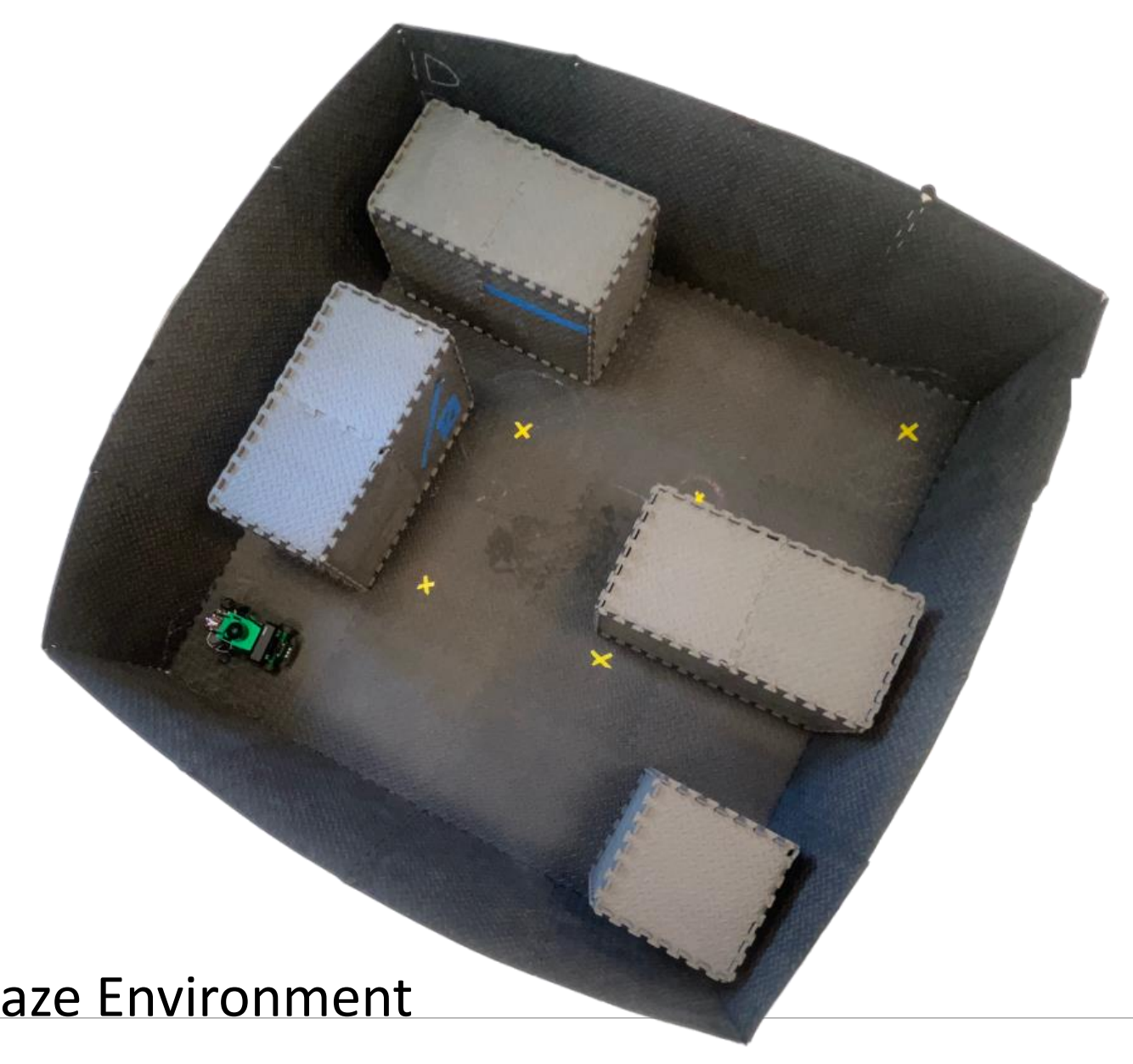
- Classified and solved four sub-problems to design our autonomous navigator

[4]



- Coded in Python, C++, XML
- ROSMASTER R2 Vehicle, ROS [3]

ROS Master R2 Robot

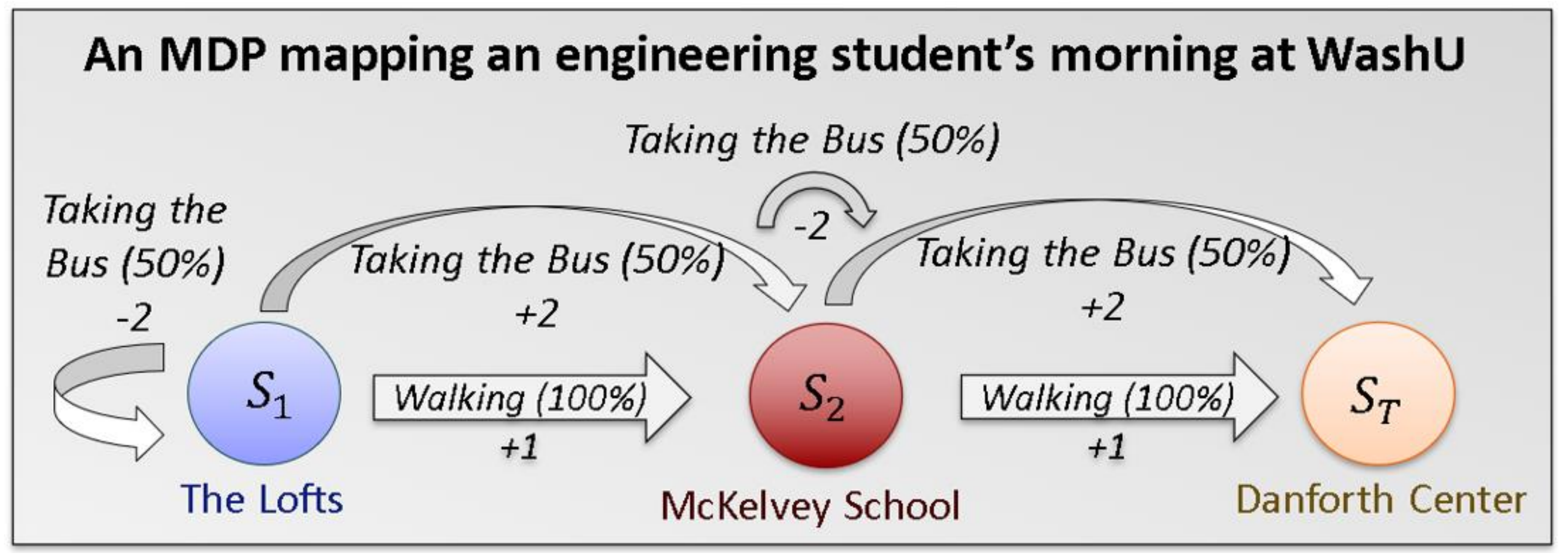

Maze Environment

3

## Markov Decision Processes

- Markov Decision Processes (MDPs) model an environment, an agent, actions and a reward value for those actions [2]
- MDPs form the basis of reinforcement learning, and we simplified our maze environment into an 8x8 MDP state space

A simple MDP where there's a 50% chance a student misses the bus



4

## Reinforcement Learning

We use RL to generate a Global Planner, or a path, that the robot will follow assuming perfect conditions. We implemented two classic RL algorithms: SARSA and Q-Learning [2]

*SARSA Update Equation*

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\right]$$

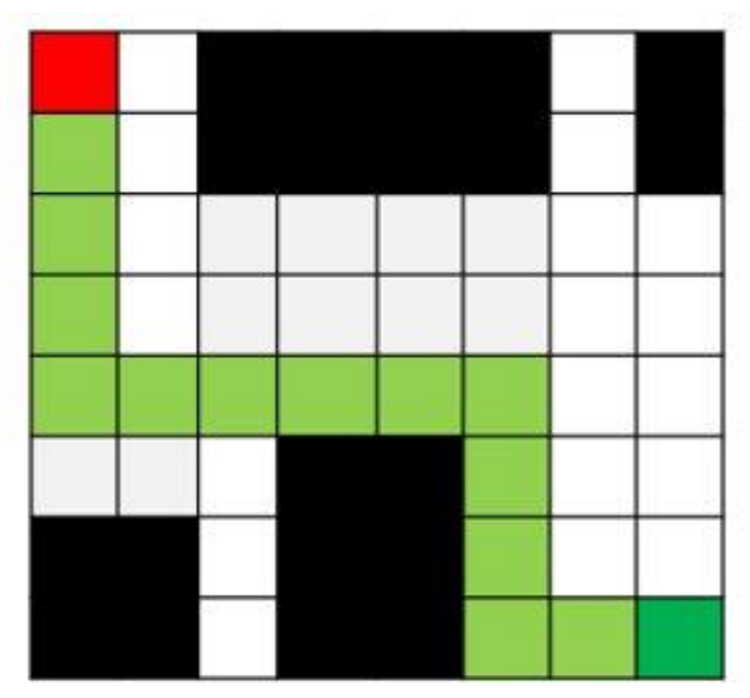
*Q-Learning Update Equation*

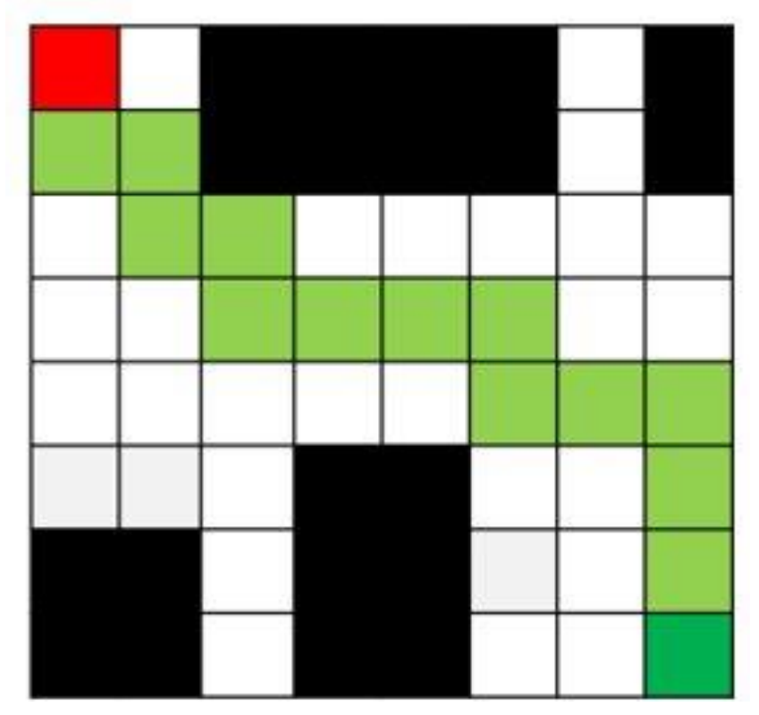
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right]$$

5

## Simulation Results

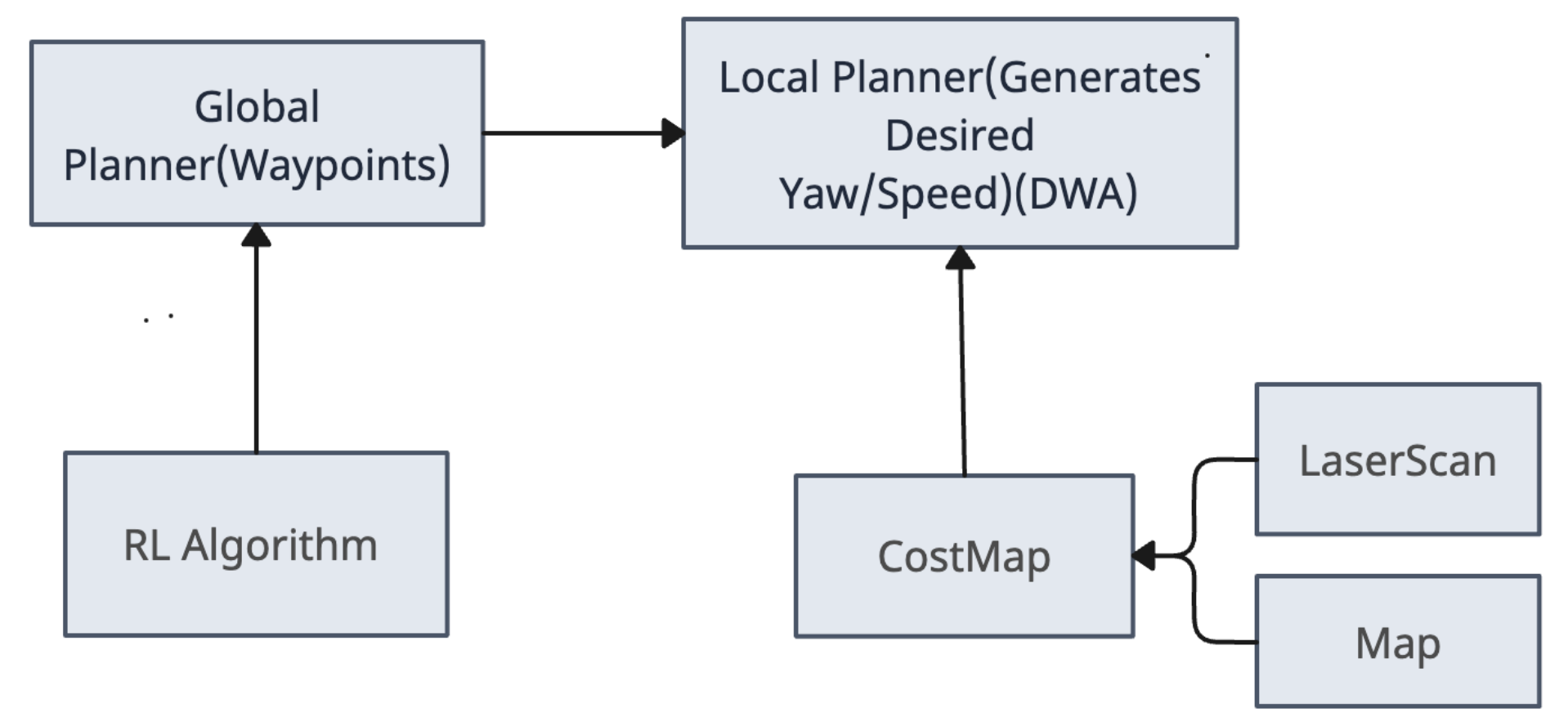
We trained both algorithms over 500 episodes with a decrementing epsilon value starting at 0.1. Below are the two policies that were created for our grid maze model.





## Global and Local Planner

*Local Planner*$[x, y, \theta, V] = (1,1,90^\circ, 1_{mps}), (1,2,90^\circ, 1_{mps}), (2,2,180^\circ, 1_{mps})$ ....

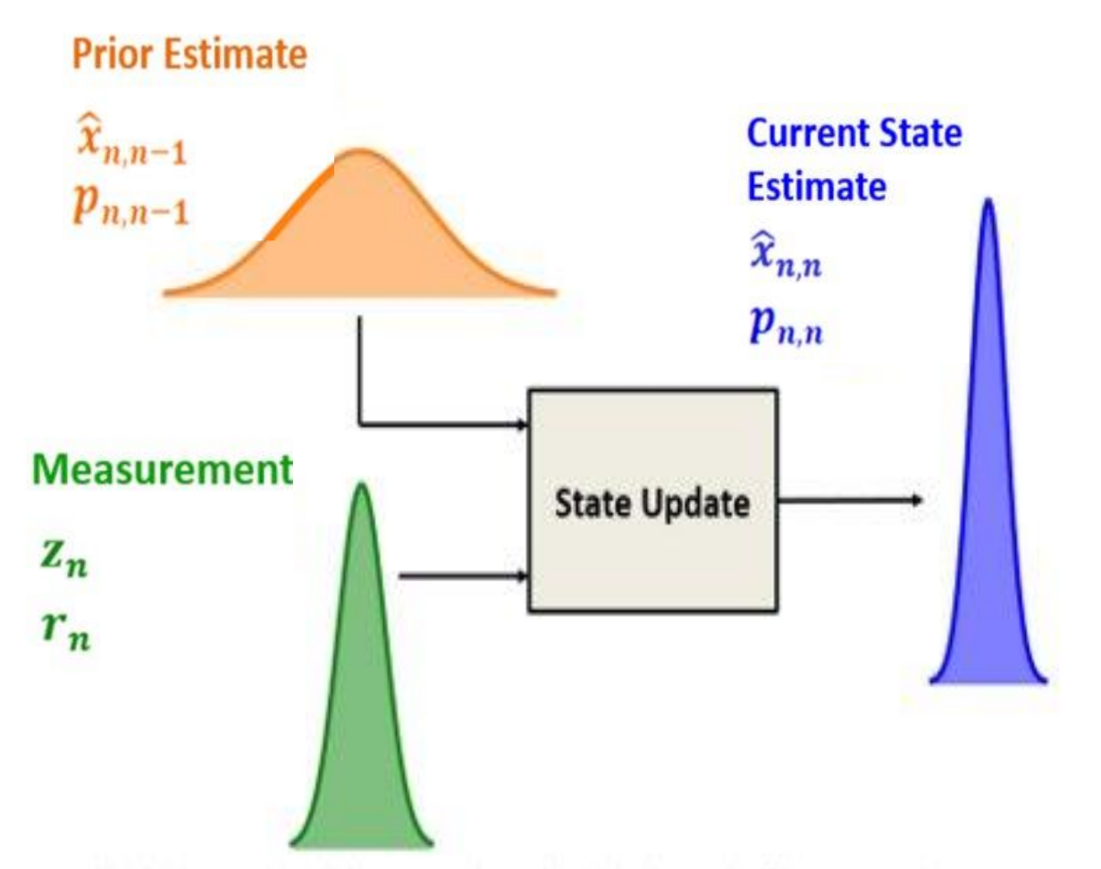

Global Planner(Waypoints)
Local Planner(Generates Desired Yaw/Speed)(DWA)
RL Algorithm
CostMap
LaserScan
Map

- RL algorithm generates global plan from map in waypoints
- Local Planner generates servo and motor commands and determines optimal position velocity and yaw

6

## Localization

- *StateVector* = $[X_t, Y_t, \theta_t, X'_t, Y'_t, \theta'_t, X''_t, Y''_t, \theta''_t]$
- Kalman filter receives odometer/IMU data
- Odometer data used for velocity
- IMU data used for position
- Weighted combination of state space model and measurement to produce state estimate [5]



Time Update ("Predict")

1. Extrapolate the state
$$\hat{\boldsymbol{x}}_{n+1,n} = \boldsymbol{F}\hat{\boldsymbol{x}}_{n,n} + \boldsymbol{G}\boldsymbol{u}_n$$
2. Extrapolate uncertainty
$$\boldsymbol{P}_{n+1,n} = \boldsymbol{F}\boldsymbol{P}_{n,n}\boldsymbol{F}^T + \boldsymbol{Q}$$

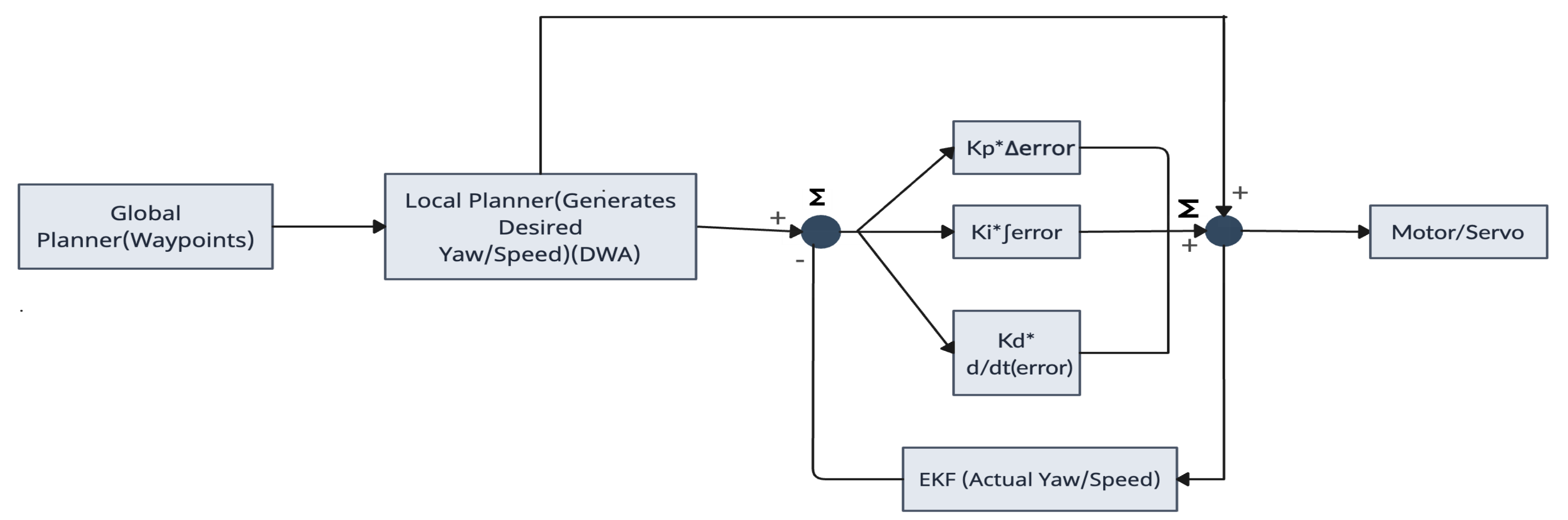
Measurement Update ("Correct")

1. Compute the Kalman Gain
$$\boldsymbol{K}_n = \boldsymbol{P}_{n,n-1}\boldsymbol{H}^T\left(\boldsymbol{H}\boldsymbol{P}_{n,n-1}\boldsymbol{H}^T + \boldsymbol{R}_n\right)^{-1}$$
2. Update estimate with measurement
$$\hat{\boldsymbol{x}}_{n,n} = \hat{\boldsymbol{x}}_{n,n-1} + \boldsymbol{K}_n\left(\boldsymbol{z}_n - \boldsymbol{H}\hat{\boldsymbol{x}}_{n,n-1}\right)$$
3. Update the estimate uncertainty
$$\boldsymbol{P}_{n,n} = (\boldsymbol{I} - \boldsymbol{K}_n\boldsymbol{H})\boldsymbol{P}_{n,n-1}(\boldsymbol{I} - \boldsymbol{K}_n\boldsymbol{H})^T + \boldsymbol{K}_n\boldsymbol{R}_n\boldsymbol{K}_n^T$$

7

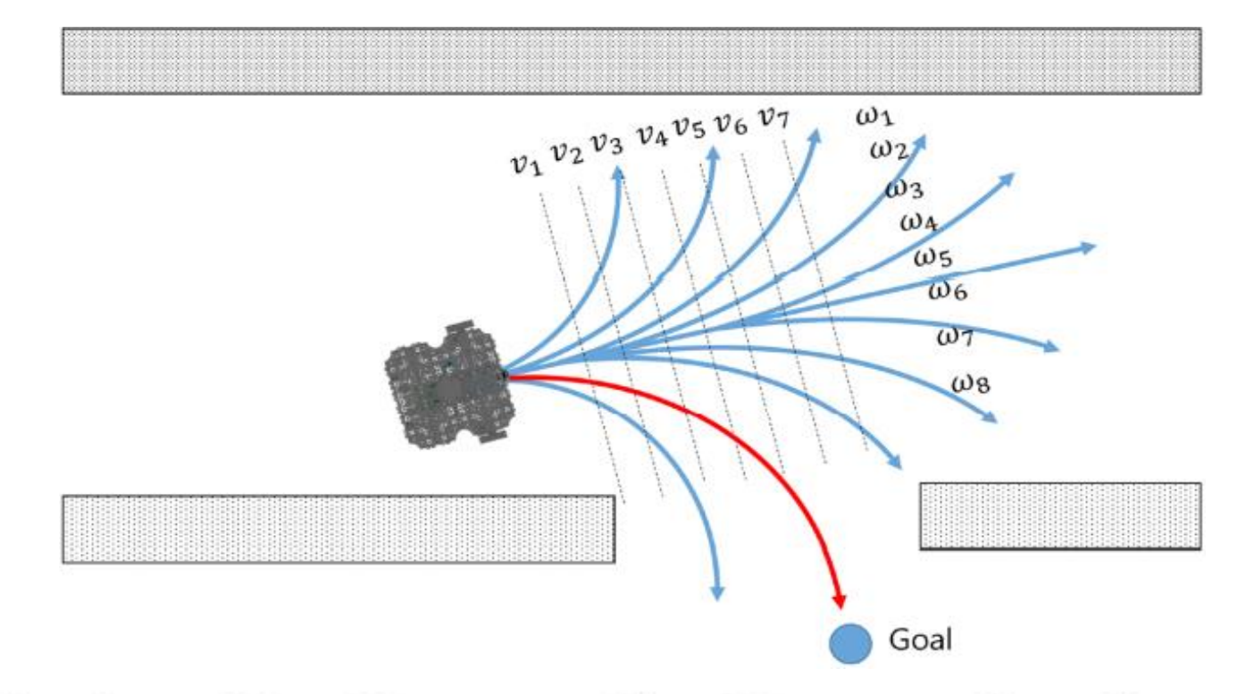
## Control

$$(Position, Velcocity) = PreValue + (Kp * error) + (Ki * Sum_{error}) + (Kd * Deriv_{error})$$


Global Planner(Waypoints)
Local Planner(Generates Desired Yaw/Speed)(DWA)
Σ
+
-
Kp*Δerror
Ki*∫error
Kd* d/dt(error)
Σ
+
+
Motor/Servo
EKF (Actual Yaw/Speed)

## Dynamic Window Approach

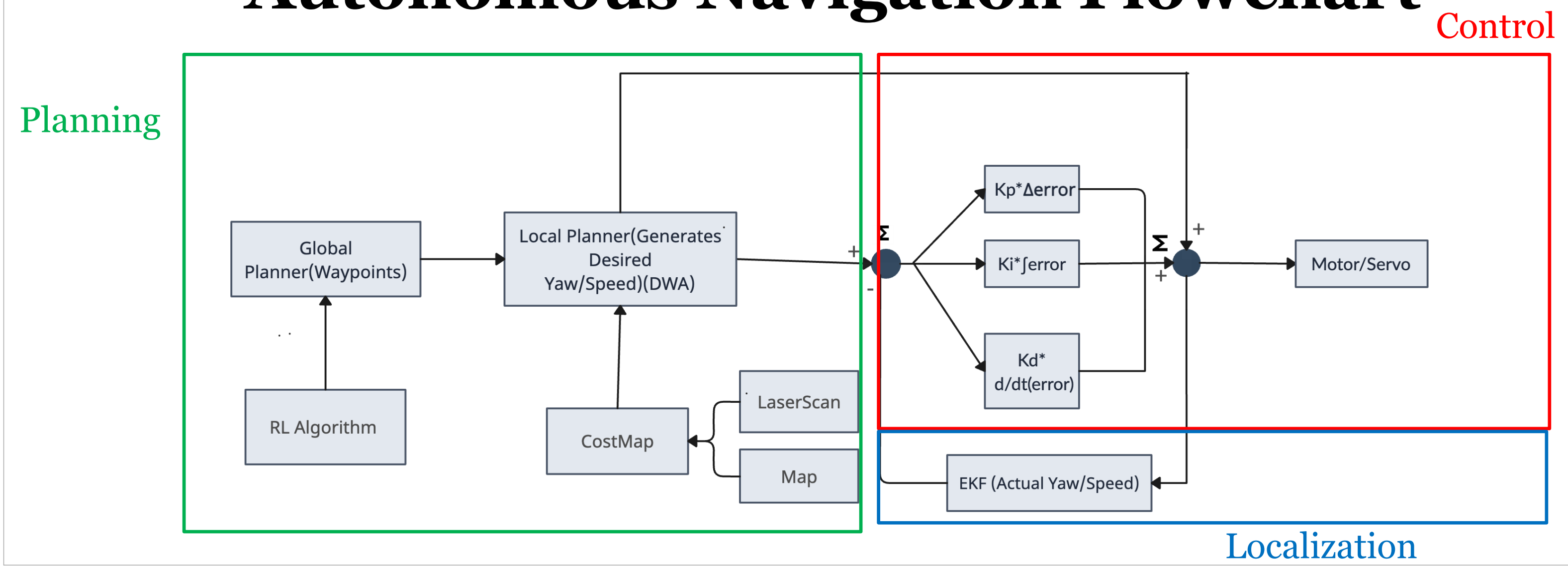
1. Sample dx, dy, dθ in the robot's control space
2. Perform forward simulation on each sampled velocity
3. Score each trajectory
4. Pick highest-scoring trajectory and send to robot [1]



$$Cost = P \cdot D_{path} + G \cdot D_{goal} + O \cdot C_{max}$$

8

## Autonomous Navigation Flowchart



9

## Results

SARSA was computationally expensive for high episode counts, indicating a need for modern learning methods. Implementing SARSA optimized between safety and efficiency across testing. Our PID controller was unable to dynamically avoid obstacles, while the DWA Planner responded dynamically to obstacles even when they were not previously indicated in the occupancy map. When combined with the PID controller, the DWA Planner proved effective in generating optimal local planner commands.

## Future Work

Improve our model's efficiency with a Deep Q-learning neural network, which should enable more sophisticated decision-making. Training our RL algorithm on a cost map will allow us to better encode the robot's understanding of its environment and subsequently improve maze navigation. To tackle uncertainties in real-world scenarios, our efforts will also include the implementation of dynamic global retraining with obstacle detection with more emphasis on parameter tuning.

## References

[1] C. Ovuegbe, "Parameter Selection in the Dynamic Window Approach Robot Collision Avoidance Algorithm using Bayesian Optimization," 2020.
[2] R. S. Sutton and A. G. Barto, "Reinforcement Learning: An Introduction," 2020.
[3] Yahboom Technology, "Rosmaster-R2," GitHub Repository, 2023.
[4] Think Autonomous, "Self-Driving Cars and Localization," [Online]. Available: https://www.thinkautonomous.ai/blog/self-driving-cars-and-localization/
[5] Kalman Filter, "Multivariate Kalman Filter," [Online]. Available: https://www.kalmanfilter.net/multiSummary.html.