

# ENGS 147 - Mechatronics Project Rat - Final Deck

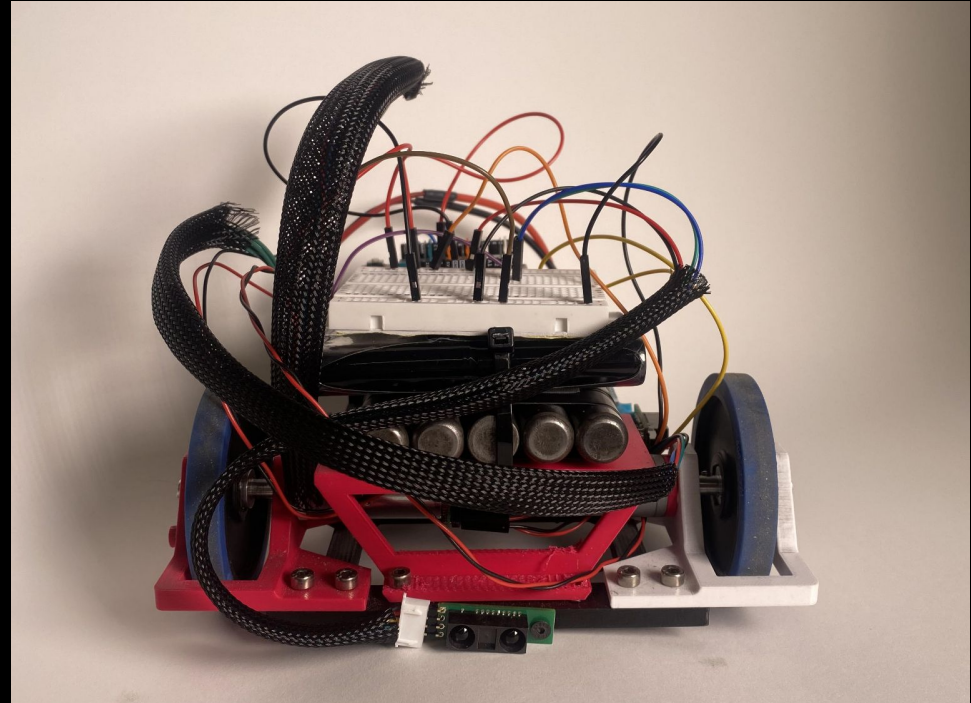
Nick, Majd, Alex, Youssef



DARTMOUTH  
ENGINEERING  
THAYER SCHOOL

# CONTENTS

- 1) Introduction
- 2) Hardware
- 3) Low Level Control
  - a) Motor Characterization
  - b) Sensor Characterization
  - c) Speed Control
    - i) Wall following
  - d) Position Control
    - i) Turning
    - ii) Stopping
- 4) Maze Algorithm
- 5) State Machine System Architecture
- 6) Code Organization

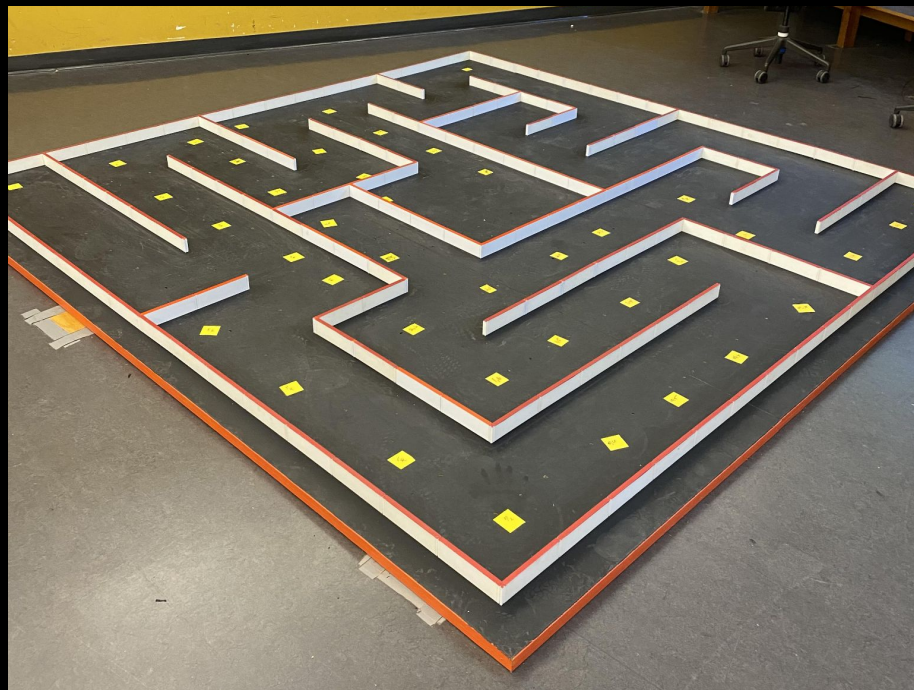


# Introduction

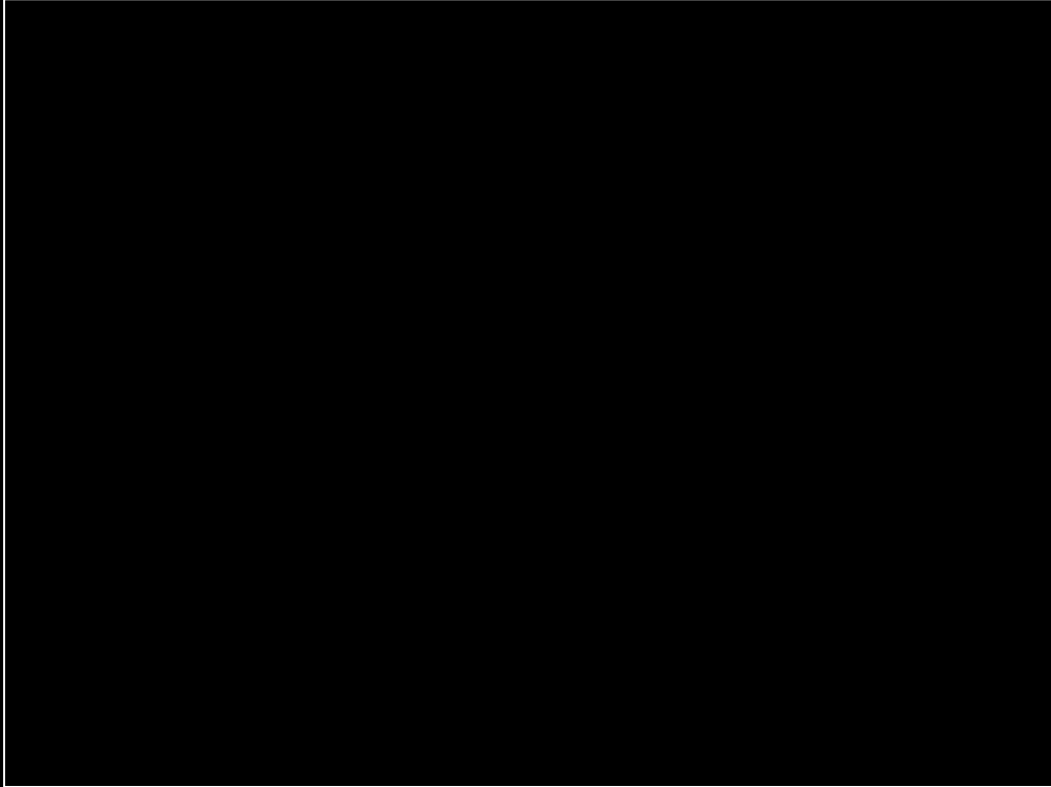
The purpose of this assignment was to design, build, and program an autonomous vehicle to navigate and escape a small maze.

Hardware consisted of off the shelf and custom fabricated components. IR proximity sensors and magnetic motor encoders were used to sense movement around the maze.

Low level speed and position controllers were designed in MATLAB and implemented on a Arduino Due.



# Our Final Submission



# Hardware - Motor Selection

$$F_{resist} = \frac{M_{mice} * G * \mu_k}{N_{motors}}$$

Take:

$m = 1.25 \text{ kg}$  (weighed previous mice)

$N = 2$

$\mu_k = 0.1$

$$\Rightarrow F_{resist} = 0.92 \text{ N}$$

Wheel radius = 0.035m

$$\tau_{req} = F_{resist} * R_{wheel}$$
$$\Rightarrow \tau_{req} = 0.0322 \text{ Nm} = 0.3883 \text{ kg cm}$$

We want to spec our motor such that our operating conditions are approximately 15-20% of the stall torque. To find the appropriate motor, we take our  $\tau_{req}$  to be 20% of the motor spec. that we choose in Polulu.

$$\tau_{motor} = \tau_{req} * 5 = 1.93 \text{ kgcm}$$

Where the small polulu motors run at 12V and we only have 9.6V battery packs we rescale the motor requirement with  $V_{new}/V_{old}$ .

$$\tau_{motor} = 1.93 \text{ kg cm} / 0.8 = 2.42 \text{ kgcm}$$

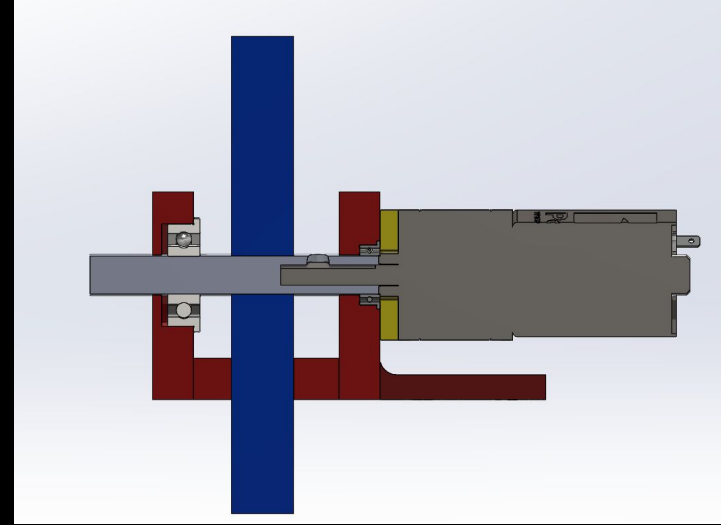
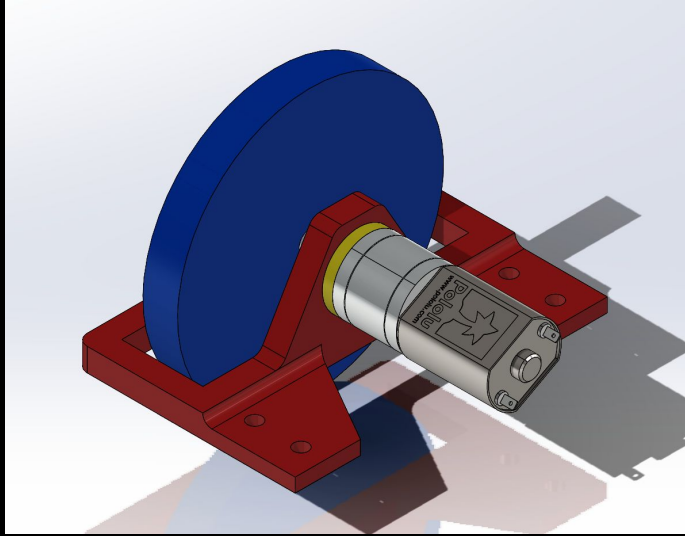
The following design specifications were considered in motor selection:

- Torque
- Operating Speed
- Size

The methodology to the left shows how torque was determined. Ultimately, a motor was chosen with a S.F ~ 2. The logic behind the higher gearing was reduce the operating speed to roughly 0.5 m/s

**Hindsight:** Prioritizing a small form factor reduces the risk of wall collision, and increases the margin of error compensators can operate under.

# Hardware - Motor Mounting



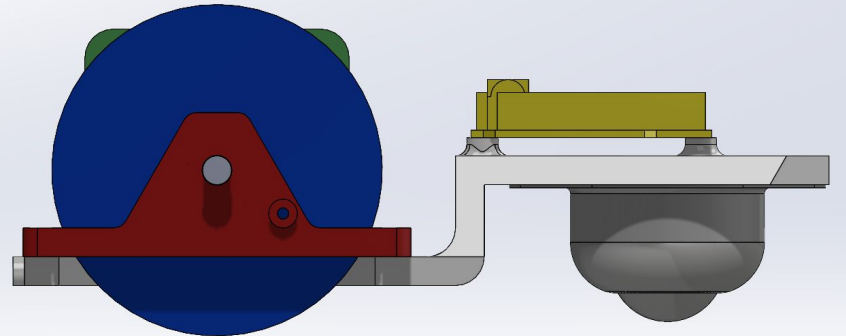
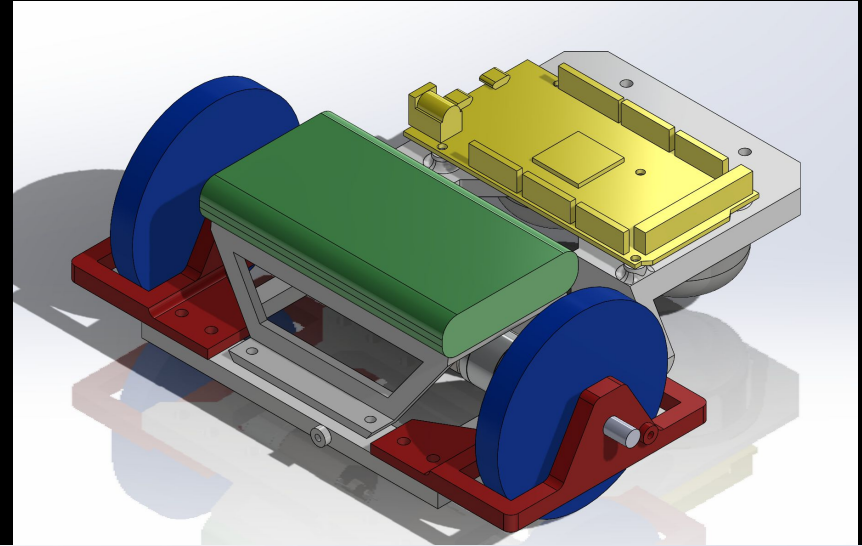
- Motors are mounted in modular packages. The drive shaft is supported by two bearings in order to reduce radial load on the motor shaft.
- Motor shaft is custom 4mm -  $\frac{1}{4}$ " to reduce overall chassis width.
- Motor velocity/ position read by shaft mounted magnetic encoders

# Hardware - Chassis

The chassis was 3D printed, and designed to be a modular such that iterations of motor units could be quickly swapped in/out.

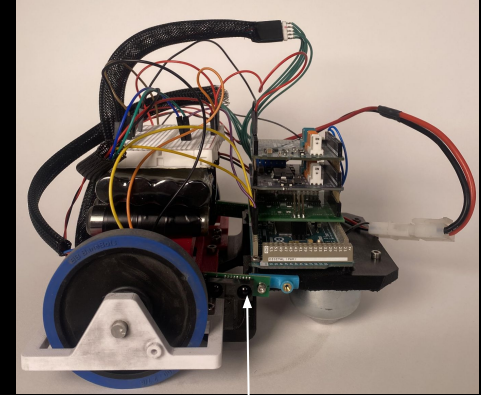
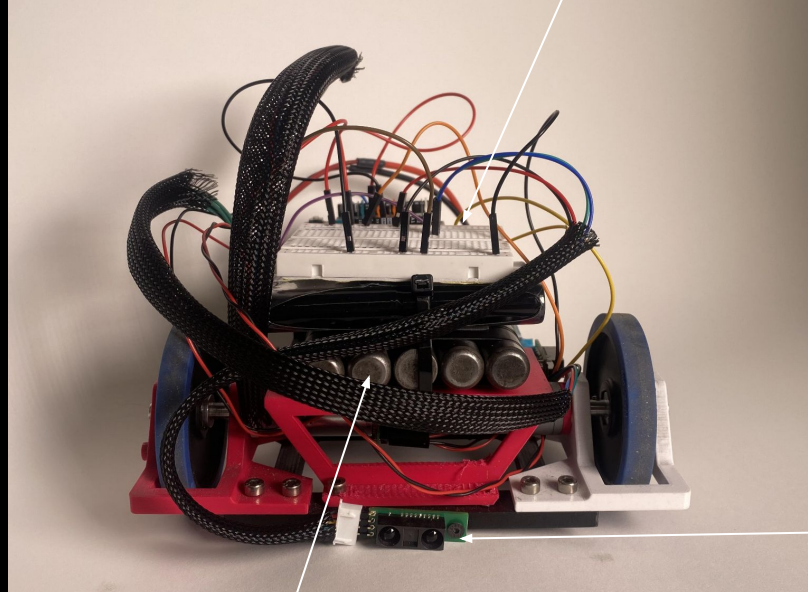
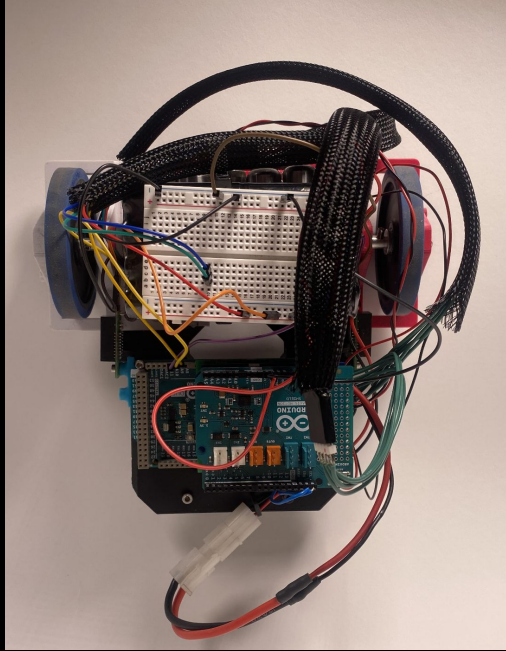
The chassis is supported by off the shelf casters.

**Hindsight:** Smaller casters could be chosen to reduce the overall dimension of the mouse, and potentially reduce friction in the system.





# Hardware - Execution



Sensor breadboard

Deadweight to improve wheel traction

3x IR Sensor

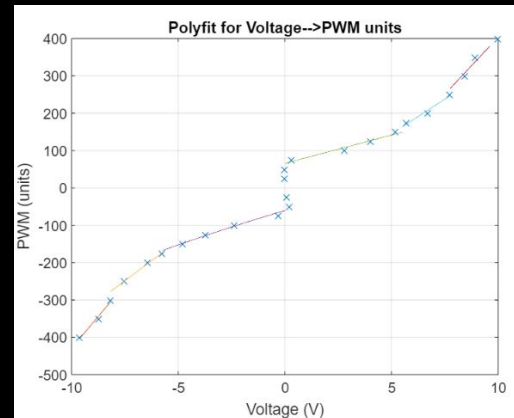
**Hindsight:** wheel slipping proved to be a significant issue which our system sensing was not able to capture. Wheel selection could be optimized for traction.



# Low Level Control - Motor Characterization

Command	Voltage (V)	Km (rad/s/unit)	Km (rad/s/V)	$\tau$ (sec)	rad/s
-400	-9.62	0.04	1.79	-0.013	-17.25
-350	-8.71	0.04	1.77	0.022	-15.46
-300	-8.18	0.05	1.77	0.04	-14.46
-250	-7.52	0.05	1.76	0.057	-13.25
-200	-6.43	0.06	1.78	0.07	-11.47
-175	-5.75	0.06	1.78	0.088	-10.23
-150	-4.78	0.06	1.77	0.093	-8.47
-125	-3.72	0.06	1.88	0.115	-6.98
-100	-2.35	0.04	1.91	0.14	-4.48
-75	-0.31	0.00	0.00	0	0
-50	0.21	0.00	0.00	0	0
-25	0.1	0.00	0.00	0	0
25	0	0.00	0.00	0	0
50	0	0.00	0.00	0	0
75	0.31	0.00	0.00	0	0
100	2.8	0.04	1.52	0.165	4.25
125	4.02	0.05	1.61	0.11	6.48
150	5.18	0.06	1.63	0.093	8.48
175	5.7	0.06	1.79	0.076	10.21
200	6.7	0.06	1.71	0.068	11.47
250	7.72	0.05	1.74	0.055	13.46
300	8.43	0.05	1.71	0.04	14.45
350	8.93	0.04	1.73	0.023	15.45
400	10	0.04	1.74	0.014	17.44

- Input PWM commands to measure voltage output and motor speed
- Measured  $\tau$  values for different s.s. speeds
- Ran s.s. Theoretical vs. Experimental tests to calibrate our  $K_m$  value to be more representative of the motor.



# Low Level Control - Sensor Characterization

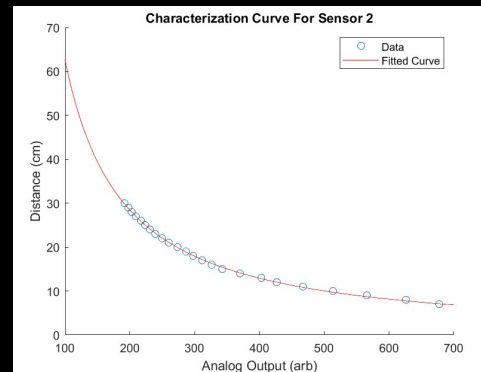
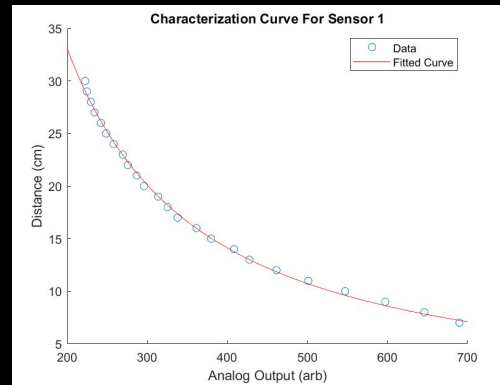
## Procedure:

- Move IR to a distance measured with ruler
- Take 100 measurements, find median (more robust to outliers)
- Repeat for all distances from 1 to 30
- Extrapolate using power law - save separate coefficients for each sensor

```
class PositionSensor
{
public:
    PositionSensor(int pin, float *coefficients);
    ~PositionSensor();
    float readDistanceCM();
    int readDistanceRaw();

private:
    int _pin;
    float *_coefficients;
};
```

Max output = 700 ~~ min distance = 6.8 cm  
Distance =  $a * (\text{output})^b$  (power law)



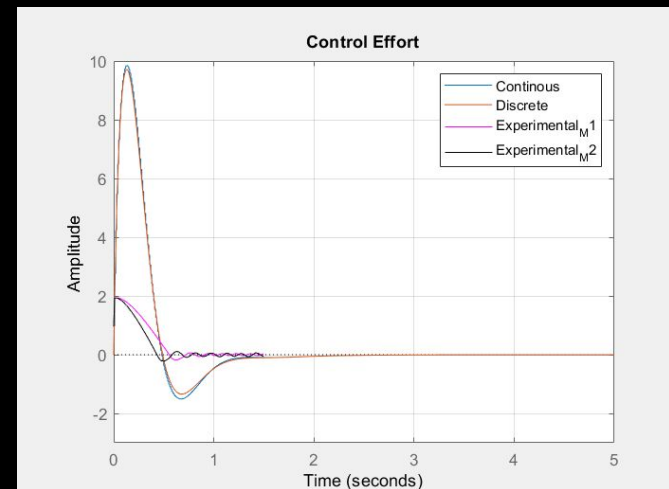
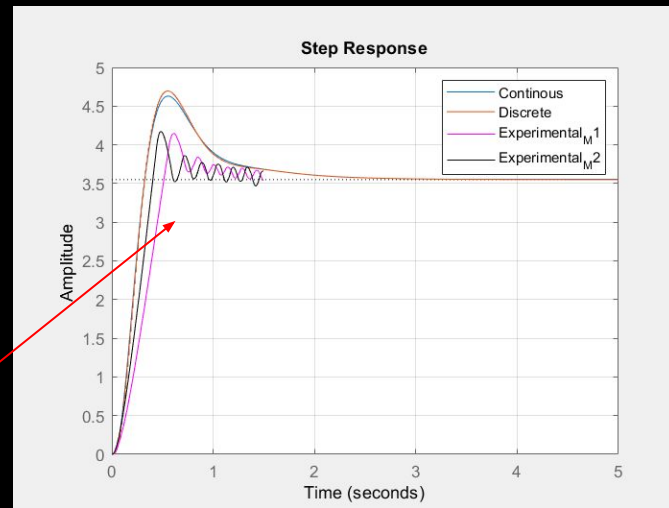
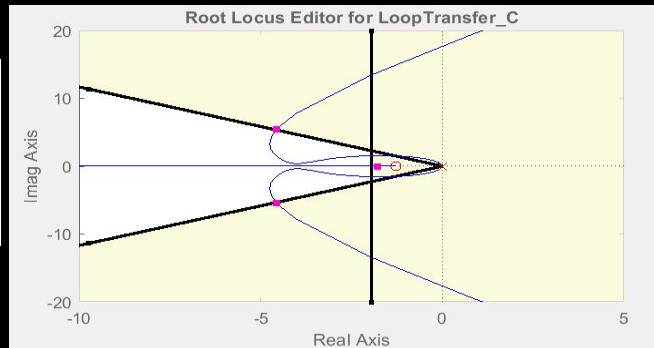
<https://github.com/Alex-Carney/Micromouse/tree/master/lib/PositionSensor>

# Low Level Control - Position

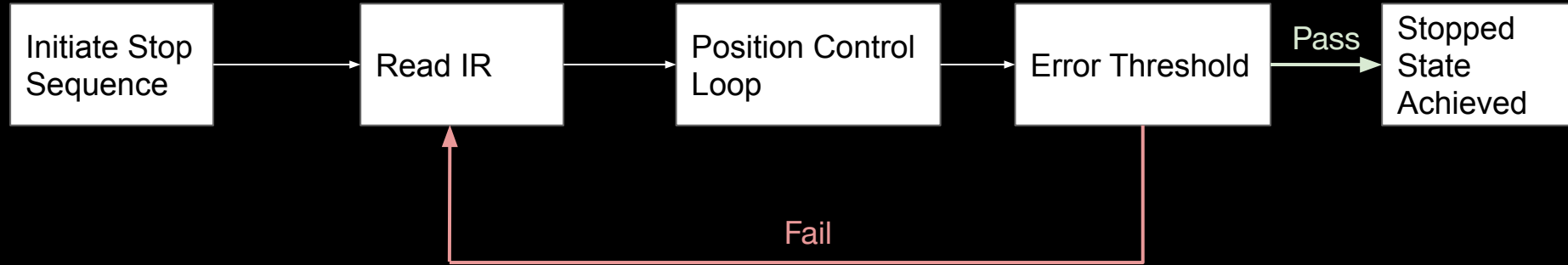
- Lead- PI designed in via Discrete Approximation in MATLAB's siso tools
- Damping prioritized ( $\sim .707$ ) with a settling time spec of  $< 2$  s
- Dithering was implemented in GetPWM function to overcome motor stiction in the event of overshoot.
- Position control exits when error threshold for both motors is met
- Sampling rate 10ms. Dither frequency at this sampling rate

ans =

$$\frac{0.2745 z^2 + 0.003491 z - 0.271}{z^2 - 1.818 z + 0.8182}$$



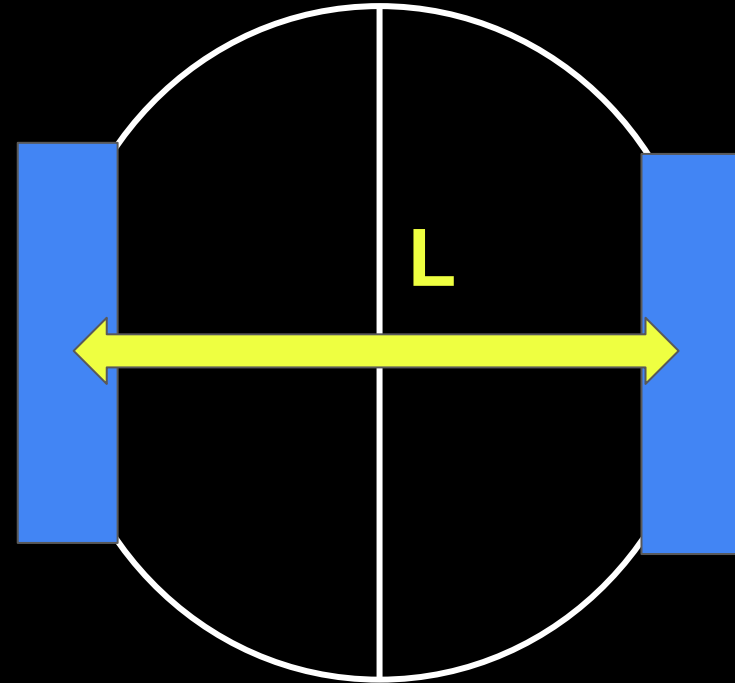
# LLC: Position Control Applied - Stopping



1. Front IR sensors detect wall node
2. Distance passed into position control and compared to reference for node alignment
3. IR sensor read again and compared to reference
4. If some disturbance entered system (bad IR reading, slip) control loop is executed with updated IR reading. Otherwise, mouse continues

# LLC: Position Control Applied - Turning

- To turn 90 degrees we need each wheel to travel  $\frac{1}{4}$  of the circle
  - $L$  = distance between wheels = 16.5 cm
  - Arc distance per wheel =  $\pi L/4 = 12.96$  cm
- Reference to wheel
  - Rotate wheel amount corresponding to that arc distance
  - Percent of a wheel rotation expressed in rads
    - $((\pi L/4)/C) * 2\pi = ((\pi L/4)/\pi d) * 2\pi$
    - Wheel ref =  $L\pi/2d$
  - $(16.5 \text{ cm} * \pi)/(2 * 7 \text{ cm}) = 3.7$  rads
- Right vs left turning reference adjusted based off

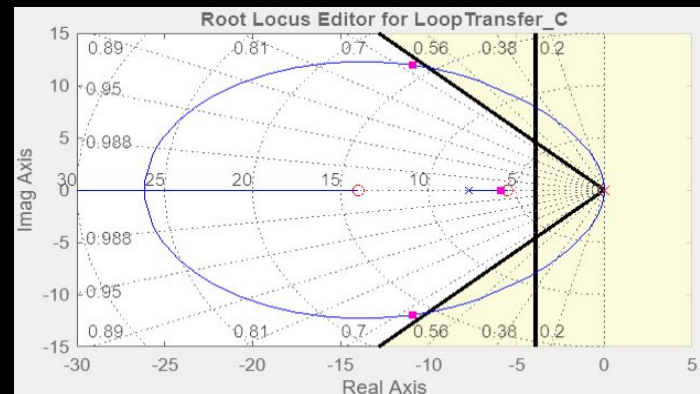
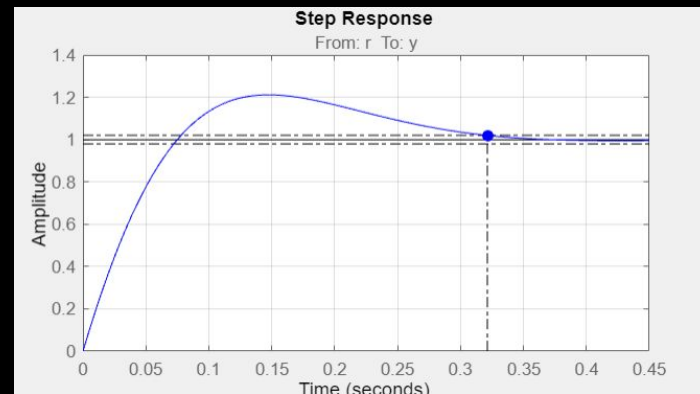
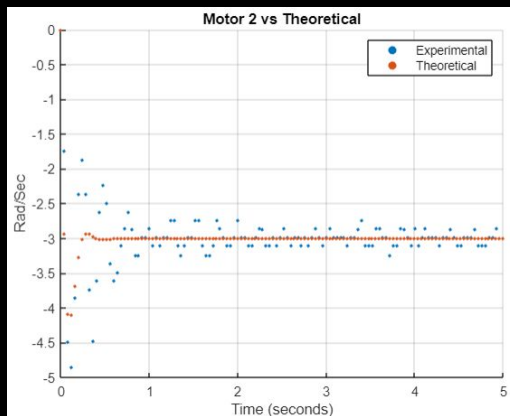
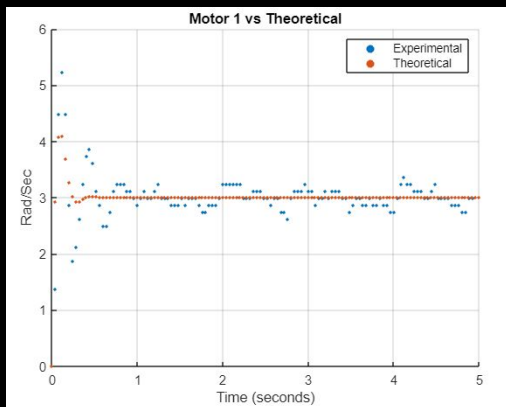


# Low Level Control - Velocity Control 3 rad/s

- Designed a controller via Discrete Approximation for constant velocity of the mouse.
- $e_{ss} = 0$  w/ disturbance.
- 40 ms sampling time

Tunable Block  
Name: C  
Sample Time: 0  
Value:  
 $1.5 (s+14) (s+5.5)$   
-----  
 $s^2$

Gcz =  
$$\frac{2.131 z^2 - 2.908 z + 0.9612}{z^2 - 2 z + 1}$$
  
Sample time: 0.04 seconds  
Discrete-time transfer function.  
[Model Properties](#)

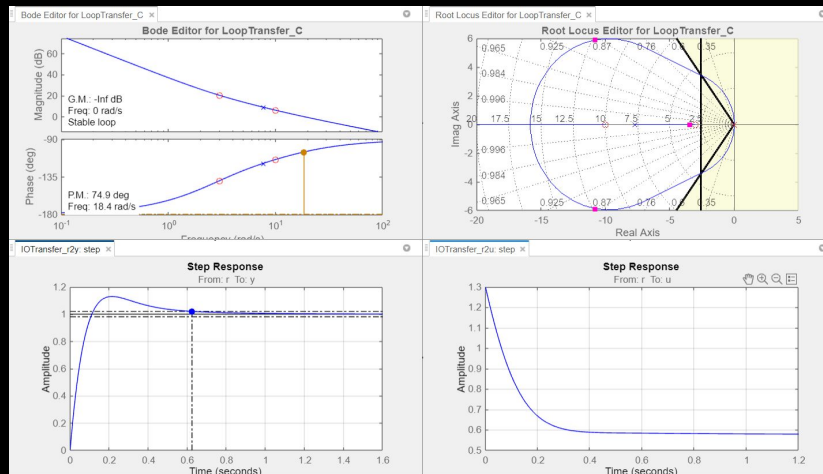


# LLC: Velocity Control 7 rad/s

- Ramp up speed
  - Higher top speed
    - Change controller for new reference
- Wall following
  - Use PD to change speed reference
  - Decrease sample time of velocity control to follow the wall better
    - 20 ms

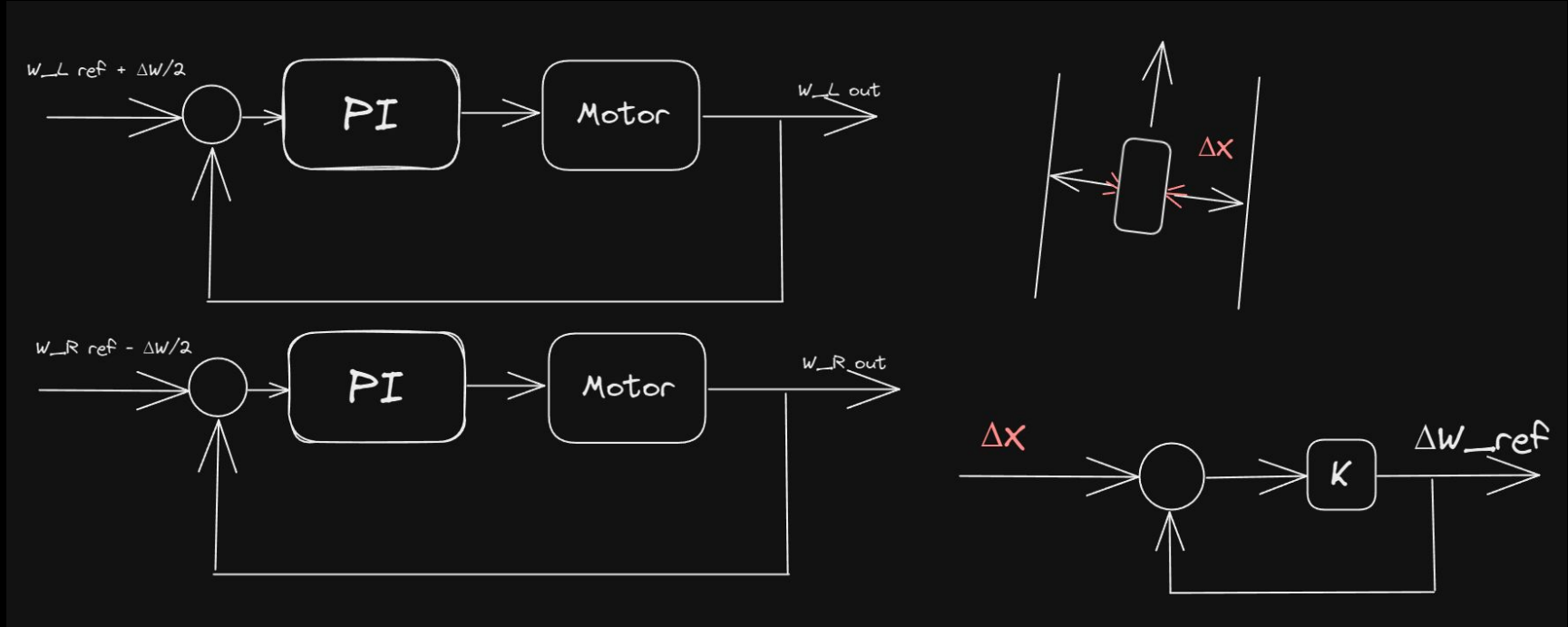
Tunable Block  
Name: C  
Sample Time: 0  
Value:  
$$\frac{1.3 (s+10) (s+3)}{s^2}$$

GcZ =  
$$\frac{1.246 z^2 - 2.193 z + 0.9603}{z^2 - 2 z + 1}$$
  
Sample time: 0.02 seconds  
Discrete-time transfer function.  
[Model Properties](#)





# LLC: Wall Following - System

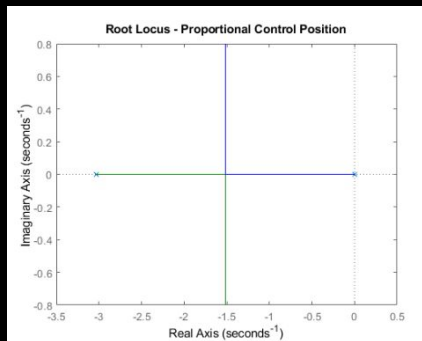


- What compensator should we use to convert  $\Delta x$  to  $\Delta w$ ? Is Proportional gain enough?

# LLC: Wall Following - Compensation

## Attempt 1: Proportional Control

- Fundamental problem: Position control with proportional gain has this root locus:



- High gain = massive oscillations
- Low gain = cannot compensate fast enough

## Attempt 2: PID Control

- High derivative gain causes system to have fast transient response
- Integrator gain removes steady state error, better response for longer walls
- Allows for larger proportional gain without oscillations
- Saturation protection: protects against derivative control effort

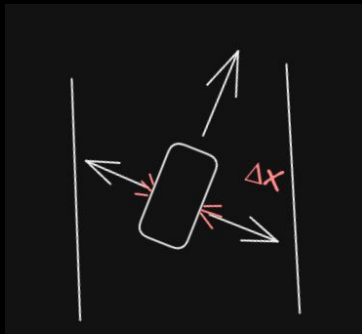
```
// Gains:  
const double Kp = 0.1;  
const double Ki = 0.01;  
const double Kd = 1;
```

```
// if del W ref is bigger than 5 set it to 5  
if (abs(delWRef) > wall_control::DEL_REF_TOO_HIGH)  
{  
    delWRef = wall_control::DEL_REF_TOO_HIGH;  
}
```

# LLC: Wall Following - Limitations and Improvements

## Limitations

- If mouse is angled, IR sensors read improperly
- Wall following while in an intersection = system failure.
- Using timing to turn on wall following after entering an intersection



```
// Check if we should turn on wall following
if (wall_control::control_based_on_wall == false)
{
    if ((newTime -
        wall_control::time_started_driving)
        >
        wall_control::WALL_FOLLOW_TIME)
    {
        wall_control::control_based_on_wall = true;
    }
}
```

```
SPEED_MODE == 1
const long WALL_FOLLOW_TIME = 1650000;
se
const long WALL_FOLLOW_TIME = 2200000;
dif
```

## Future Improvements

- Use NAxisShield heading to correct IR sensor distances (project position vector onto "x" axis)
- When traveling through an intersection, use 1 wall to follow instead of 0
- Replace timing control with position or logical control

# Maze Algorithm-Traversal

- Packaged in an Arduino Library: MazeTraversal.h, MazeTraversal.cpp
- Called every time the mouse reaches an intersection
- Expects 5 integers indicating whether the mouse can go right, left, forward or backward and the right motor encoder reading (To find the distance the mouse moved since last intersection)
- Implements DFS to traverse the maze using a stack:
  - Once the mouse reaches an intersection, the possible directions are pushed to the stack and the direction on the top of the stack is marked as visited
  - Once we reach an intersection where multiple paths can be taken, all paths are pushed to the stack
  - If the mouse reaches a dead end, we enter the backtracking state where we pop the top of the stack and instruct the mouse to move in the opposite direction of the top of the stack

1 1 0 1	Stack:	Right, visited
0 1 1 1		
0 0 0 1		
1 1 0 1	Stack:	Left, visited Right, not visited Left, visited Right, visited
0 1 1 1		
0 0 0 1		
1 1 0 1	Stack:	Right, not visited Left, visited Right, visited
0 1 1 1		
0 0 0 1		
1 1 0 1	Stack:	Right, visited Left, visited Right, visited
0 1 1 1		
0 0 0 1		

# Maze Algorithm-Traversal

- To void possible cycles and ensure correct mapping from the physical map to an array representing the maze, we only push cardinal direction on the stack where North is taken as initial direction the mouse is pointing
- Conversion between cardinal directions and mouse direction is performed within the MazeTraversal library using:

```
○ int convertToMouseOrientation(int d);  
○ struct Path convertToMazeOrientation(int path_right, int  
path_left, int path_forward, int path_back);
```

```
1 1 0 1  
0 1 1 1  
0 0 0 1
```

Stack: East, visited

```
1 1 0 1  
0 1 1 1  
0 0 0 1
```

Stack: West, visited  
East, not visited  
North, visited  
East, visited

```
1 1 0 1  
0 1 1 1  
0 0 0 1
```

Stack: East, not visited  
North, visited  
East, visited

```
1 1 0 1  
0 1 1 1  
0 0 0 1
```

Stack: East, visited  
North, visited  
East, visited

# Maze Algorithm-Mapping

- The stack method is not sufficient to identify a cycle if the maze contains one
- To detect cycles, we map the number of steps the mouse moves in one direction to 2-D array
- We find the number of steps from the encoder reading where  $1000 = 1$  step
- 1 indicated visited squares and 0 indicated unvisited squares
- No changes are made to the visited array in the backtracking state
- If we write to a non-zero element in the 2-D array while in the traversing state, then the mouse is moving in a path that it has explored before.
- Once a cycle is detected, we enter the backtracking state and traverse back to the first non-explored element of the stack

1	1	1	1	
0	1	1	1	Stack:
0	0	0	1	West, visited
				East, not visited
				North, visited
				East, visited

1	1	1	1	
0	1	1	1	Stack:
0	0	0	1	South, visited
				West, visited
				East, not visited
				North, visited
				East, visited

Cycle detected, backtrack

1	1	1	1	
0	1	1	1	Stack:
0	0	0	1	East, visited
				North, visited
				East, visited

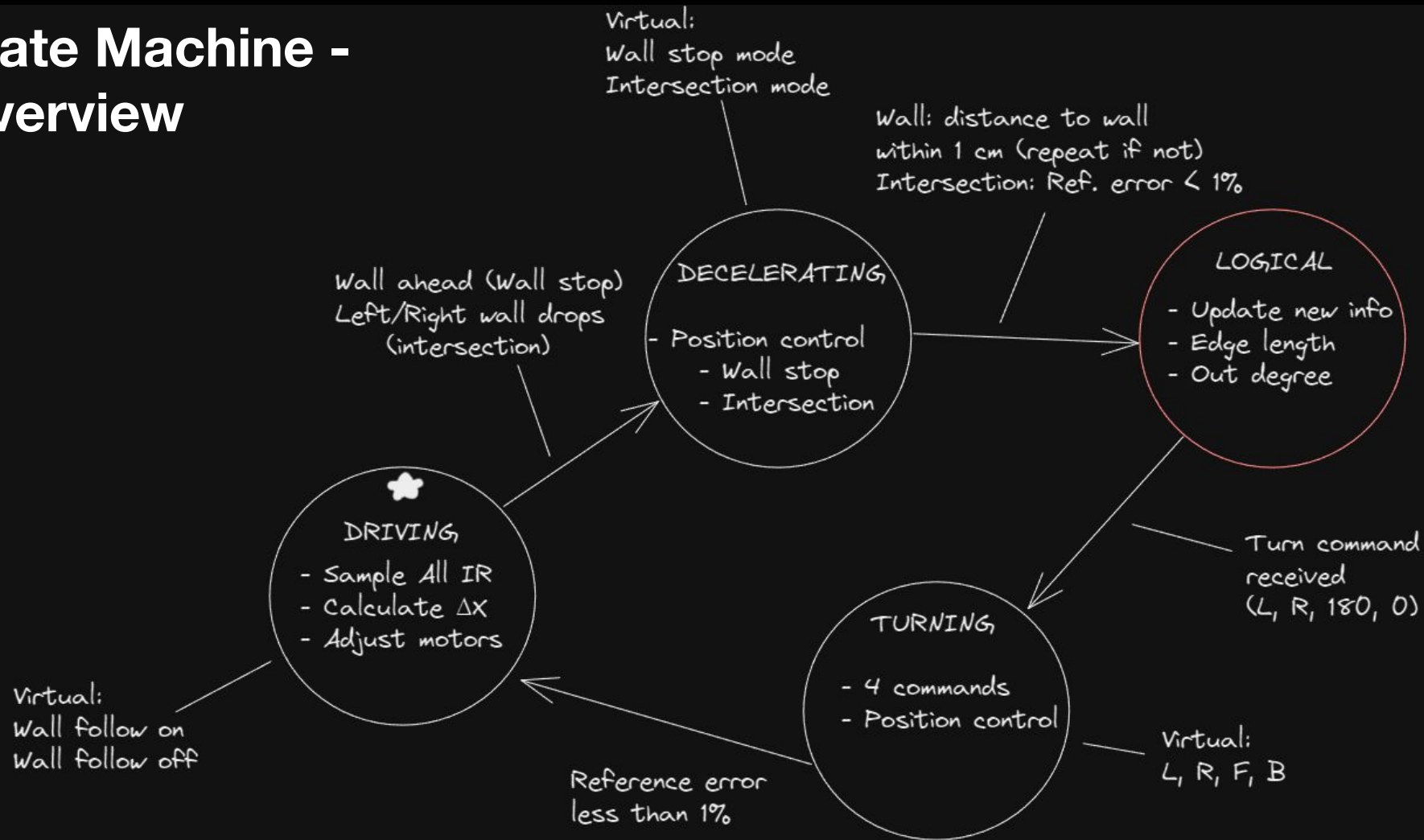
# Maze Algorithm-Ending

- Given the encoder to cells mapping, the maze ending appears as 3x3 square on the visited array where the edges are non-zero and the center is zero
- Every time the mouse detects an intersection, it checks for such square in the visited array. If found, the mouse stops traversing the maze

```
1 1 1
1 0 1
1 1 1
```



# State Machine - Overview



# State Machine - Virtual States

- Different actions within the same state, based on logic determined by previous state
- Avoids clutter of states (1 turning state rather than 4, 1 deceleration state instead of 2, etc.)

```
if (pos_control::stop_mode == pos_control::StopMode::WALL_PRESENT)
{
    // Compensate distance based on wall
    pos_control::position_reference = pos_to_move_rad;
}
else
{
    // No wall to compensate for
    pos_control::position_reference =
        (traversal::INTERSECTION_WALK_DISTANCE * 2 * M_PI)
        / WHEEL_CIRCUMFERENCE;
}
```

- StopMode triggers between 2 types of decelerating states
- No need for 2 separate states entirely

```
= turning::TurnDirection::LEFT;

    if (turning::turn_count == 1)
    {
        = turning::TurnDirection::RIGHT;

        if (turning::turn_count == 1)
        {
            = turning::TurnDirection::FORWARD;

            if (turning::turn_count == 1)
            {
                = turning::TurnDirection::BACKWARD;
```

- TurnDirection triggers between 4 types of turning states

# State Machine - Implementation + Code Organization

```
switch (MACHINE_STATE)
{
case MachineState::INITIALIZING:
case MachineState::DRIVING:
case MachineState::DECELERATING:
case MachineState::LOGIC:
case MachineState::TURNING:
case MachineState::PROCESS_STOPPED:
}
```

- Finite states, all possibilities accounted for in switch statement

```
> namespace traversal...

> namespace pos_control...

// Global variables for DRIVING

> namespace drive_control...

> namespace wall_control...

// Global variables for TURNING

> namespace turning...
```

- Proper scoping of all parameters required for each state (i.e: drive\_control::SAMPLE\_TIME or pos\_control::SAMPLE\_TIME)

```
pos_control::stop_mode = wall ?
    pos_control::StopMode::WALL_PRESENT :
    pos_control::StopMode::NO_WALL;
MACHINE_STATE = MachineState::DECELERATING;
isFirstStateIteration = true;
break; // Cancel this state early! Don't continue
```

- State transitions can be performed at any point in state
- All states have an “initialization” stage - isFirstStateIteration

# Software Architecture

```
#ifndef _MACROS_H_
#define _MACROS_H_

// macro for debug mode: 0 of
#define DEBUG_MODE 0
// macro for speed mode: 0 s
#define SPEED_MODE 1
// macro for delay between st
#define STATE_DELAY 1
#define STATE_DELAY_VALUE 50
// macro to switch between po
#define POS_MODE 1
```

- Custom macros for rapid iteration



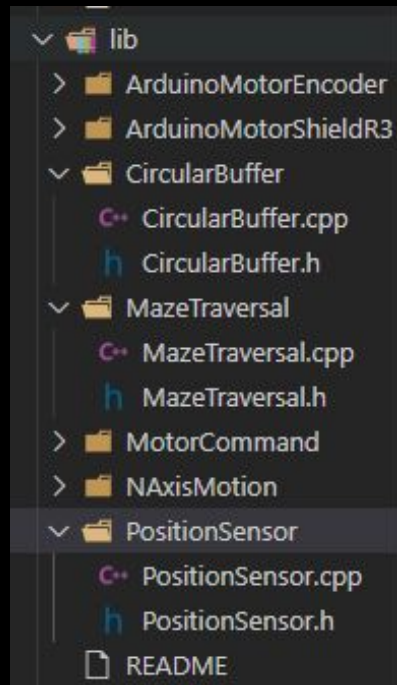
<https://github.com/Alex-Carney/Micromouse>

- Github for proper collaboration and development

```
// Free used memory
void MazeTraversal::Clean(){
    for (int i = 0; i < numRows; i++) {
        delete[] visited[i];
    }
    delete[] visited;
    delete[] stack;
}

~CircularBuffer()
{
    // Deallocate memory for the buffer
    for (int i = 0; i < steps; i++)
    {
        delete[] buffer[i];
    }
    delete[] buffer;
}
```

- Proper memory management for all libraries. Results: 0 memory/seg fault issues during entire project



- Utility code properly packaged into libraries

# Thank You

We would like to thank Professor Ray and Professor Kokko for their advice and mentorship during this process!

master


5 branches

0 tags


Go to file

Add file

<> Code


 majdh98 final video code

00e7107 3 days ago

 71 commits

folder	.vscode	basic maze traversal	last week
folder	include	This code beats the maze	5 days ago
folder	lib	final video code	3 days ago
folder	other	My changes	5 days ago
folder	src	final video code	3 days ago
folder	test	first commit	last month
file	.gitignore	first commit	last month
file	README.md	Create README.md	last month
file	platformio.ini	NAxis works now ffs	last week

☰ README.md



## Micromouse

---

### ENGs 147 - Mechatronics

---

Dartmouth College, S23

Alex Carney, Youssef Marzouk, Nicholas Hagler, Majd Hamdan