Purdue University

Skitter Robot Control

Michael Barth & Jackson Emerson

11/29/2024

Systems, Measurement, and Control II

School of Mechanical Engineering

Purdue University

585 Purdue Mall

West Lafayette, IN 47907

Table of Contents

Introduction

Throughout the semester, the lab assignments in ME 375 have helped students become familiar with essential control system concepts as well as the various subsystems of the Skitter robot. The final project provides an opportunity to apply the knowledge gained throughout the course to meet specific project objectives. The primary goal of the project is to design a controller and program the Skitter robot to operate autonomously through a series of tasks. The robot must follow a line around a track, and after completing one lap, it will switch to a different mode where it moves forward and backward to track and follows an object in front of it.

The Skitter robot consists of several key subsystems that are integral to its overall performance. These subsystems include the encoders, ultrasonic sensors, line follower sensor, motors, and the Arduino controller.

The robot used in the project is the AndyMark Skitter Classroom Robot (model AM-55150). It is equipped with various components and sensors, such as an Arduino Mega, two DC motors with encoders, a servo motor, an ultrasonic sensor, and a line follower sensor. The Arduino Mega features 54 digital input/output pins, 4 serial ports, a USB port, and a reset button. The DC motors are equipped with internal gearboxes and encoders, with each 12V motor capable of reaching up to 245 RPM and generating a maximum torque of approximately 410 g-cm. The ultrasonic sensor has a detection range of 2 cm to 400 cm and offers high accuracy (± 3 mm). The line follower sensor is able to detect dark lines within a 6 mm range. These subsystems are all essential for achieving the objectives outlined in the final project.

Michael Barth wrote the introduction and explanations of subsystem characterizations. Jackson Emerson wrote the proposed controller structure and design and potential problems.

Subsystem Characterization

The encoders play a crucial role in the system, as their data is used to accurately switch between the line-following mode and the object-following mode. The encoder output represents how far the wheels have rotated. From testing, counts per 360-degree rotation were found and tabulated in Table 1. These values can be used to estimate the distance the robot has traveled by performing some basic calculations. By knowing the encoder output, the number of rotations can be determined, and multiplying this by the wheel's circumference provides an approximate distance traveled. This information can be used to trigger the transition to line-following mode after completing a lap.

Table 1: Motor Encoder Count

Encoder	Counts per 360-degree rotation
Right	-175.6
Left	169.5

The ultrasonic sensor, located at the front of the Skitter robot, is essential for the object-following mode. It measures distance in meters. The robot needs to maintain a distance of roughly 9 inches (0.2286 meters) from an object. When the object moves farther away, the robot will move forward; if the object comes too close, the robot will reverse. To avoid jittery movements, a steady state error of 0.5 inches is implemented.

The line-following sensor provides multiple outputs that help determine the robot's position relative to the line. It features three binary outputs indicating whether the robot is to the left, on, or to the right of the line. Additionally, the sensor has a line position output that indicates how close the robot is to the center of the line, with values ranging from -1 to 1, where 0 means the robot is centered. This output offers more precise information than a typical binary output. The line-following sensor is critical for completing the first lap. The binary outputs can be used in a state machine to guide the robot's movements—deciding whether it should turn left, right, or continue straight. Through testing, the optimal motor speed ranges can be determined to maximize the robot's performance in these states.

Finally, the motor-gearbox wheels make up the last subsystem. The robot's behavior has been modeled as a first-order system, leading to the following transfer function and single plant model used for both wheels given by Equation 1. To determine the static gain and time constant, a Pulse Generator was employed with the following parameters: amplitude of 1, period of 1 second, pulse width of 50%, and a phase delay of 0 seconds. Using the data obtained from this setup, the static gain and time constant were calculated for both motors. Since the values were relatively similar, the average of the two motors' static gain and time constant was taken to create a single, unified plant model.

$$P(s) = \frac{K}{\tau s + 1} = \frac{5.07}{0.079s + 1}$$

Equation 1: Wheel Plant Model

Further testing was performed to acquire deadband values for each motor. These parameters are shown in Table 2. The upper bound of the deadband is the voltage needed for the motors to overcome the force of friction the wheels experience in the forward direction, also known as friction compensation.

Table 2: Motor Deadband Values

Motor	Deadband (V)
Right Motor	0 – 0.17
Left Motor	0 – 0.22

Proposed Controller Structure and Design

The control structure of this robotic system integrates multiple sensors, feedback loops, and controllers to achieve precise and adaptive behavior. At its core, the system employs a closed-loop control strategy to maintain stability and respond dynamically to changes in its environment. Sensors, such as the line and ultrasonic sensors, continuously monitor the robot's surroundings, while encoders track the velocity and position of its wheels. This data is processed through the Arduino-based control unit, which orchestrates the decision-making process via a state machine.

Feedback from the encoders enables the calculation of real-time wheel velocities, which are compared to the target velocities defined by the state machine. These discrepancies are minimized by PID controllers, ensuring that the robot accurately executes the desired motions. Friction compensation further refines control by adjusting for mechanical losses, ensuring smooth and reliable operation. The control structure, as illustrated in the schematic above, highlights the interconnected roles of sensing, decision-making, and actuation, which collectively govern the robot's behavior.

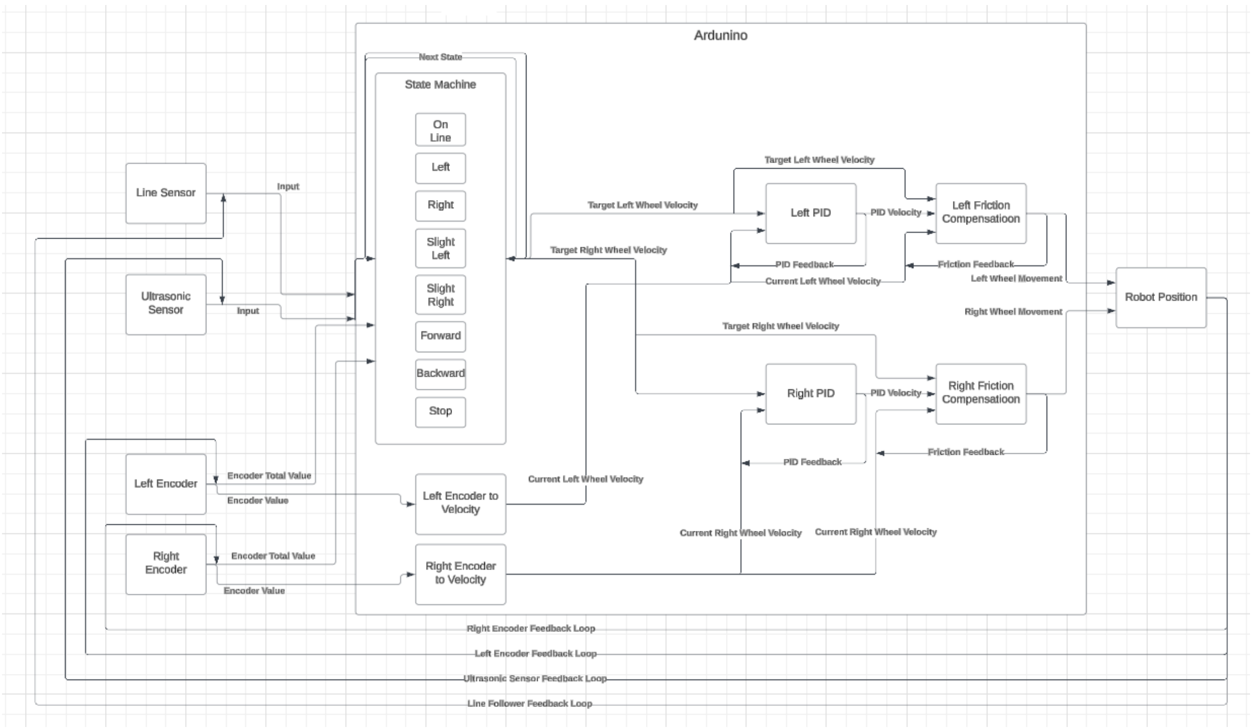

Ardunino
Next State
State Machine
On Line
Left
Right
Slight Left
Slight Right
Forward
Backward
Stop
Line Sensor
Input
Ultrasonic Sensor
Input
Left Encoder
Encoder Total Value
Encoder Value
Right Encoder
Encoder Total Value
Encoder Value
Left Encoder to Velocity
Right Encoder to Velocity
Current Left Wheel Velocity
Target Left Wheel Velocity
Target Right Wheel Velocity
Target Left Wheel Velocity
Left PID
PID Velocity
Left Friction Compensatioon
PID Feedback
Friction Feedback
Current Left Wheel Velocity
Left Wheel Movement
Right Wheel Movement
Robot Position
Target Right Wheel Velocity
Right PID
PID Velocity
Right Friction Compensatioon
PID Feedback
Friction Feedback
Current Right Wheel Velocity
Current Right Wheel Velocity
Right Encoder Feedback Loop
Left Encoder Feedback Loop
Ultrasonic Sensor Feedback Loop
Line Follower Feedback Loop

Figure 1: Control Structure Schematic

The heart of this control system lies in the state machine, which dictates the robot's actions based on sensor inputs and system feedback. Each state corresponds to a specific movement or action, such as turning, moving forward, or stopping. By transitioning between these states, the robot adapts to its environment and maintains alignment with its operational objectives. The following section delves into the state machine's design and functionality, illustrating its pivotal role in enabling autonomous control.

In line-following mode, the robot evaluates the distance it has traveled to determine whether it should remain in this mode or transition to object-tracking mode. While in line-following mode, the robot continuously monitors its position relative to the line and adjusts its trajectory accordingly. The adjustments are based on the robot's current position reading and can be classified into five distinct states. First, if the line position value falls within the range of -0.2 to 0.2, the robot continues to move straight by maintaining equal speeds on both wheels. Second, for values between -0.6 and -0.2, the robot makes a slight right adjustment by stopping the left wheel while allowing the right wheel to continue at its normal speed. Third, for values between 0.2 and 0.6, the robot makes a slight left adjustment by stopping the right wheel while the left wheel continues. Fourth, if the line position value exceeds 0.6, the robot makes a large left adjustment by reversing the right wheel at 1.2 times its normal speed while the left wheel moves forward. Lastly, if the line position value is less than -0.6, the robot makes a large right adjustment by reversing the left wheel at 1.2 times its normal speed while the right wheel moves forward.

The distinction between slight and large adjustments lies in the robot's wheel speeds. For slight adjustments, one wheel stops while the other continues, whereas for large adjustments, one wheel reverses direction to create a sharper turn. These states enable the robot to reacquire the line effectively while still maintaining its ability to drive straight when appropriate. Once the robot completes one full lap, its encoder value is compared to a predetermined threshold corresponding to the track's total distance. When the encoder value exceeds this threshold, the robot transitions from line-following mode to object-tracking mode, signifying the next phase of its operation.

In object-tracking mode, the robot uses its ultrasonic sensor to continuously monitor the distance to the closest object in front of it. The desired range is 9 inches, plus or minus 0.5 inches, meaning the robot should stop within this 1-inch range. If the distance to the object is greater than 9.5 inches, the robot moves forward until the value falls below this threshold. Conversely, if the object is closer than 8.5 inches, the robot moves backward. Once the robot positions itself within the range of 8.5 inches to 9.5 inches, it stops and maintains its position while continuing to track the object. Any deviation from this range causes the robot to adjust its position accordingly to stay within the desired distance.

For a clearer view of the robot's transitions and states, refer to Figure 2.

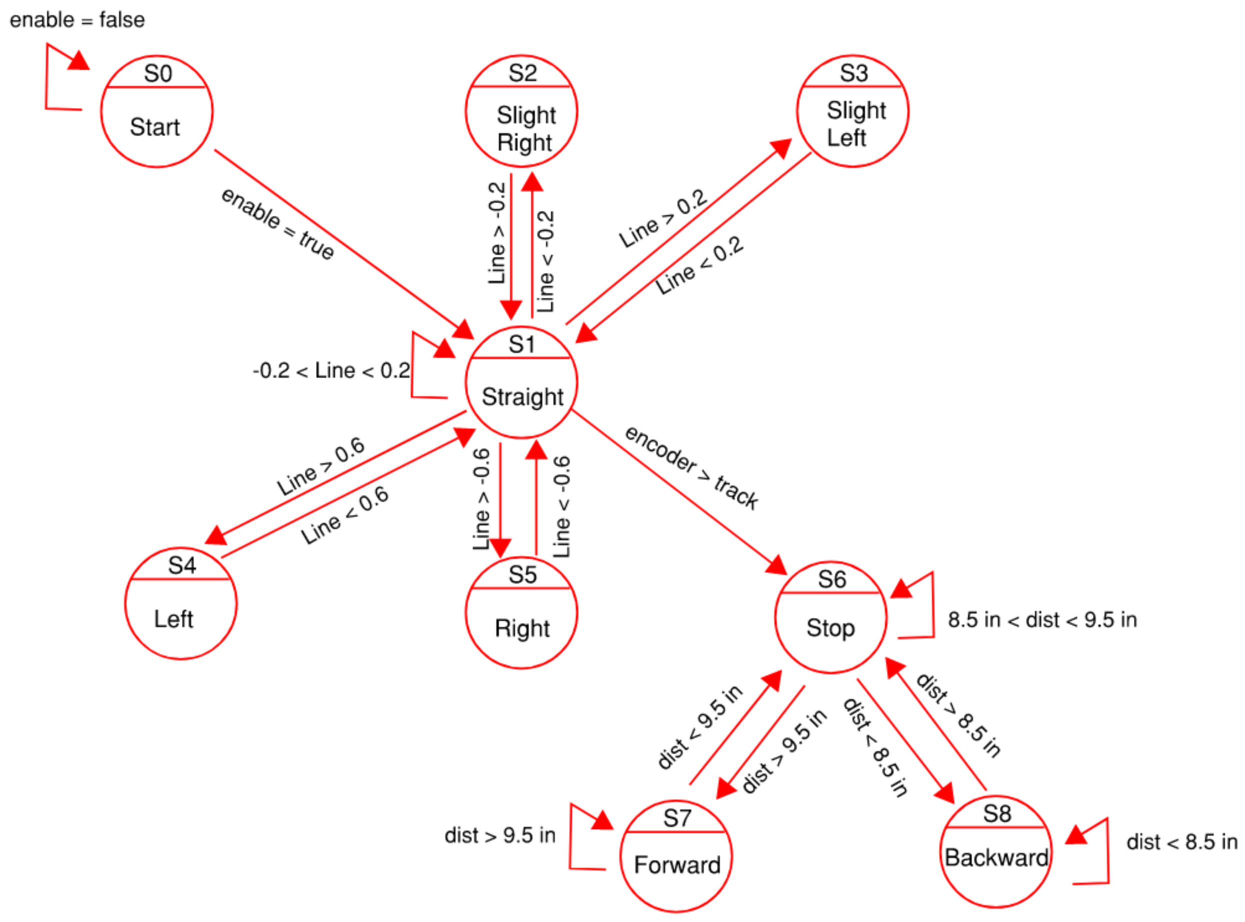


Figure 2: State Machine Diagram

The control system consists of six different feedback loops to ensure that the robot operates with optimal precision and stability. The first feedback loop is a PID controller, which is used to regulate motor speed. The PID controller was selected because of its effectiveness at handling error correction, providing smooth and responsive control by adjusting the proportional, integral, and derivate terms. The values of each for the PID being 0.4, 0.3, and 0.1 respectively, but subject to change in actual testing conditions. Friction compensation is handled by a proportional controller, which adjusts the motor speed to account for surface variations, with a gain corresponding to its deadband values, 0.17 and 0.22. Deadband values for each motor were determined by slowly incremating motor voltage, until motor rotation was achieved.

In addition, the ultrasonic sensor and line follower sensor provide continuous feedback to adjust the robot's position. The ultrasonic sensor helps maintain appropriate distance from obstacles, while the line follower sensor keeps the robot aligned with the black line. For both, a proportional gain is used of 1. Threshold values were determined by scoping outputs when the robot was on the track. Lastly, feedback from the left and right motor encoders ensures accurate tracking of the robot's position and velocity, allowing for micro-adjustments to maintain precise movement.

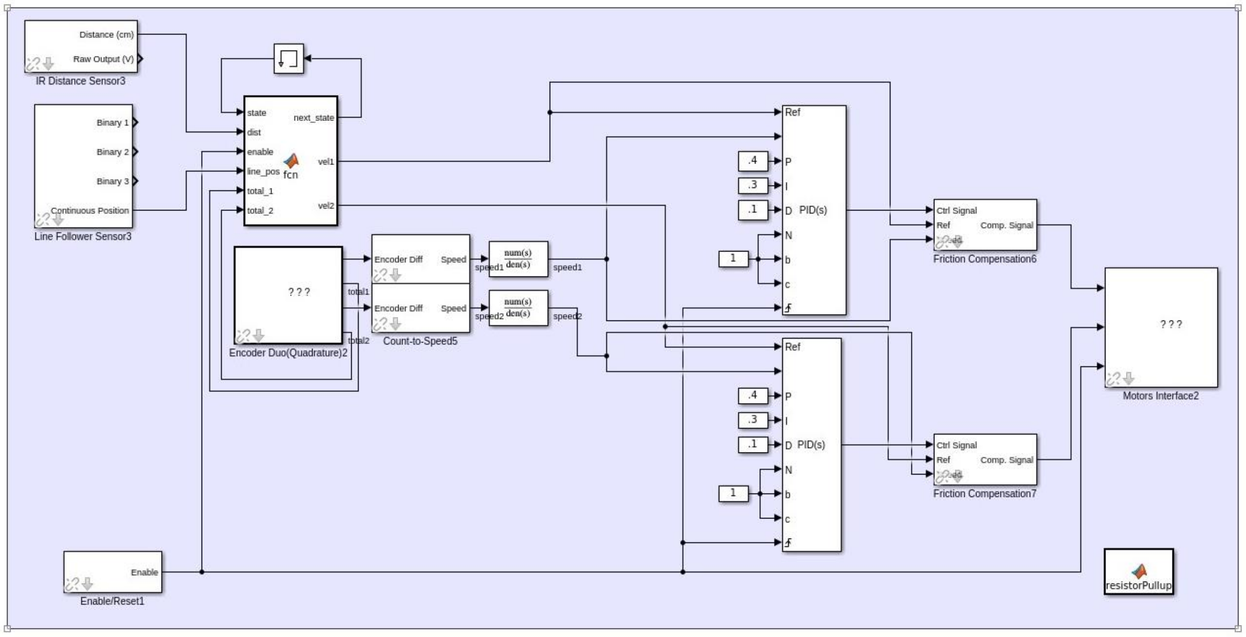
Potential Problems

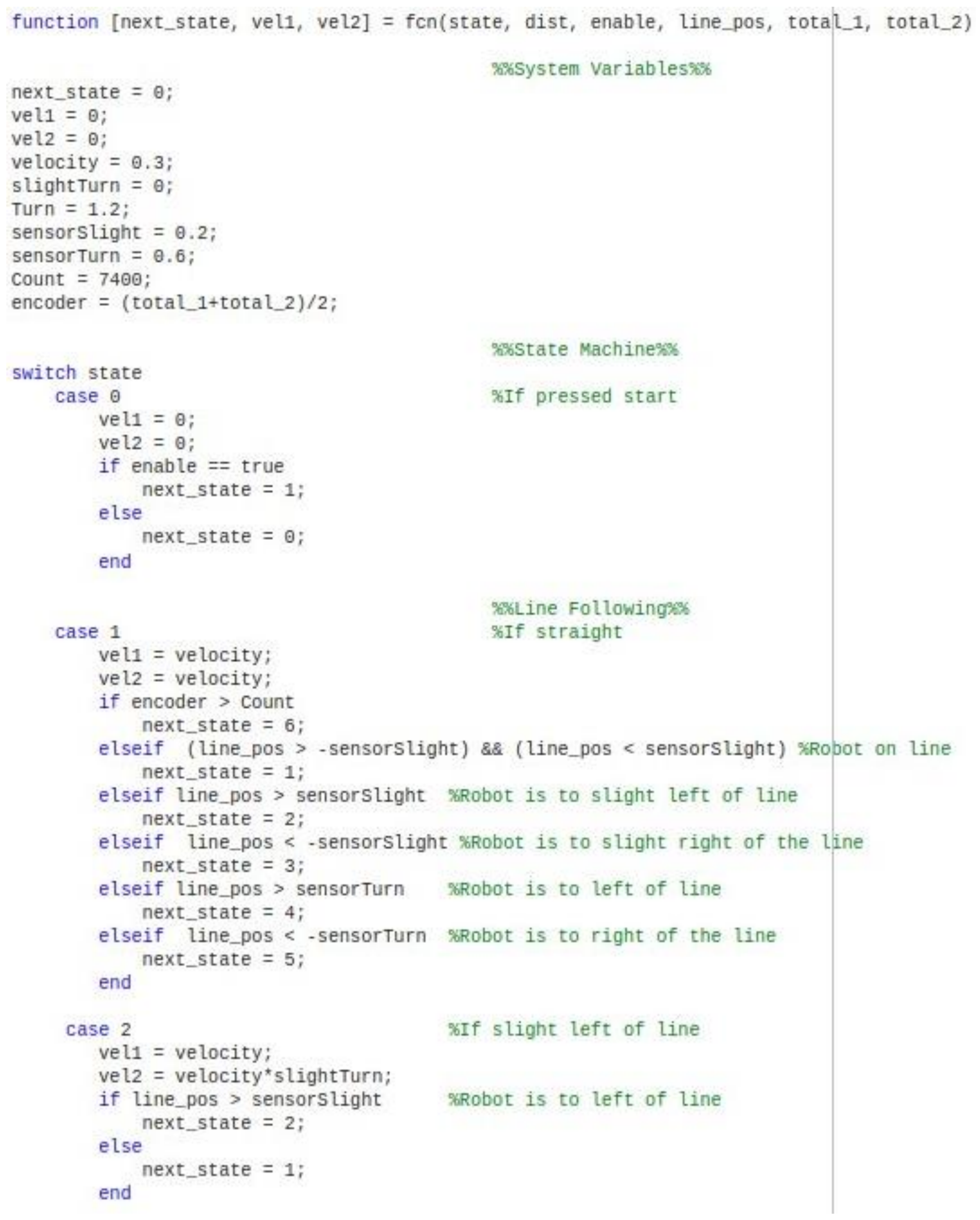
When implementing the control scheme, several challenges may arise. One of the most fundamental issues is the potential failure of the state machines to transition properly between states. This could stem from various causes, with the most likely being mis-calibrated transition limits. To address this, the system should undergo extensive testing under controlled conditions to verify that state transitions occur correctly. If discrepancies are found, micro-adjustments can be made to fine-tune the transition parameters. Additionally, it is essential to thoroughly review the code to ensure that the state machine includes all necessary states, that states are correctly labeled, and that transitions are logically defined.

A challenge with the line follower sensor is ensuring accurate interpretation of its continuous position value, which indicates the robot's alignment relative to the black line. Calibration involves testing under real conditions to map sensor outputs to positions such as centered on the line, partially off, or on the white surface. Environmental factors like lighting can affect readings, so periodic re-tuning is crucial for consistent performance. The continuous position data allows for smoother and more precise alignment adjustments. Scopes attached to the outputs of the line follower sensor blocks in Simulink will allow for testing to be performed with the robot connected to a personal laptop. Using this setup, critical information for debugging will be shown as real time sensor readings and robot response can be analyzed.

While the line follower is running, the tracking of the encoder is also taking place. However, the robot has to make micro-adjustments continuously as the robot is following the line. This can cause the encoder to have a greater value than is originally anticipated. If it is too high, then the comparison to know when it needs to transition into object tracking will occur too soon. To account for this, the average of the two motors encoder values will be taken. This will then provide a more reasonable number to track for when to transition. Additionally, slight adjustments to our threshold value corresponding to one lap can be made.

Appendix


Distance (cm)
Raw Output (V)
IR Distance Sensor3
Binary 1
Binary 2
Binary 3
Continuous Position
Line Follower Sensor3
state
dist
enable
line_pos
total_1
total_2
next_state
vel1
vel2
fcn
? ? ?
Encoder Duo(Quadrature)2
total1
total2
Encoder Diff
Speed
Count-to-Speed5
num(s)
den(s)
speed1
speed2
Ref
.4
.3
.1
1
P
I
D
N
b
c
PID(s)
Ctrl Signal
Ref
Comp. Signal
Friction Compensation6
Friction Compensation7
Motors Interface2
Enable
Enable/Reset1
resistorPullup



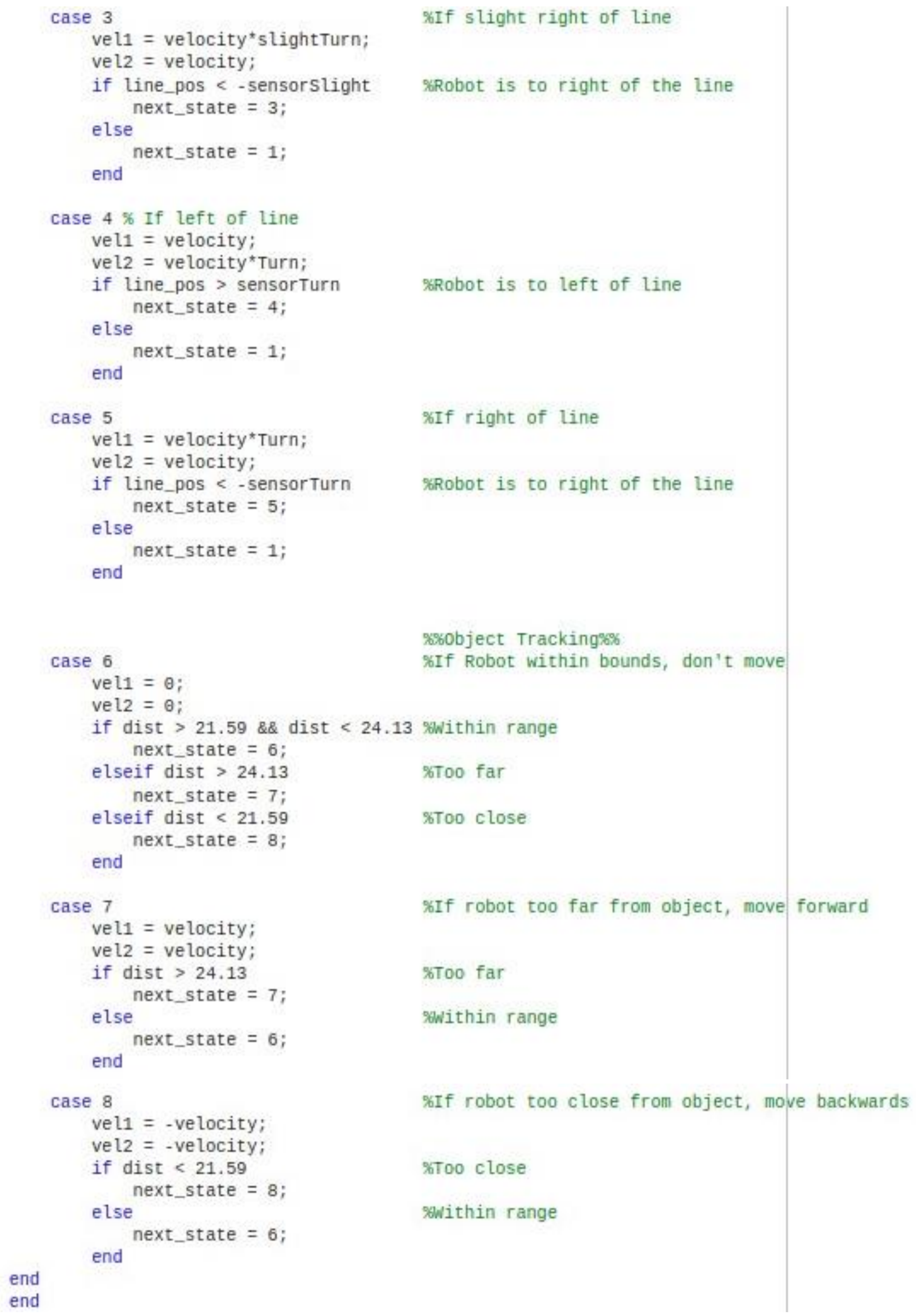
```
function [next_state, vel1, vel2] = fcn(state, dist, enable, line_pos, total_1, total_2)

                                            %%System Variables%%
next_state = 0;
vel1 = 0;
vel2 = 0;
velocity = 0.3;
slightTurn = 0;
Turn = 1.2;
sensorSlight = 0.2;
sensorTurn = 0.6;
Count = 7400;
encoder = (total_1+total_2)/2;

                                            %%State Machine%%
switch state
    case 0                                  %If pressed start
        vel1 = 0;
        vel2 = 0;
        if enable == true
            next_state = 1;
        else
            next_state = 0;
        end

                                            %%Line Following%%
    case 1                                  %If straight
        vel1 = velocity;
        vel2 = velocity;
        if encoder > Count
            next_state = 6;
        elseif  (line_pos > -sensorSlight) && (line_pos < sensorSlight) %Robot on line
            next_state = 1;
        elseif line_pos > sensorSlight  %Robot is to slight left of line
            next_state = 2;
        elseif  line_pos < -sensorSlight %Robot is to slight right of the line
            next_state = 3;
        elseif line_pos > sensorTurn    %Robot is to left of line
            next_state = 4;
        elseif  line_pos < -sensorTurn  %Robot is to right of the line
            next_state = 5;
        end

     case 2                             %If slight left of line
        vel1 = velocity;
        vel2 = velocity*slightTurn;
        if line_pos > sensorSlight      %Robot is to left of line
            next_state = 2;
        else
            next_state = 1;
        end
```



```
    case 3                              %If slight right of line
        vel1 = velocity*slightTurn;
        vel2 = velocity;
        if line_pos < -sensorSlight     %Robot is to right of the line
            next_state = 3;
        else
            next_state = 1;
        end

    case 4 % If left of line
        vel1 = velocity;
        vel2 = velocity*Turn;
        if line_pos > sensorTurn        %Robot is to left of line
            next_state = 4;
        else
            next_state = 1;
        end

    case 5                              %If right of line
        vel1 = velocity*Turn;
        vel2 = velocity;
        if line_pos < -sensorTurn       %Robot is to right of the line
            next_state = 5;
        else
            next_state = 1;
        end


                                        %%Object Tracking%%
    case 6                              %If Robot within bounds, don't move
        vel1 = 0;
        vel2 = 0;
        if dist > 21.59 && dist < 24.13 %Within range
            next_state = 6;
        elseif dist > 24.13             %Too far
            next_state = 7;
        elseif dist < 21.59             %Too close
            next_state = 8;
        end

    case 7                              %If robot too far from object, move forward
        vel1 = velocity;
        vel2 = velocity;
        if dist > 24.13                 %Too far
            next_state = 7;
        else                            %Within range
            next_state = 6;
        end

    case 8                              %If robot too close from object, move backwards
        vel1 = -velocity;
        vel2 = -velocity;
        if dist < 21.59                 %Too close
            next_state = 8;
        else                            %Within range
            next_state = 6;
        end
end
end
```