Robot design proposal

Scott Lawson 34359117

Liam Hodgson 29154119

Rowan Walsh 30817118

John Harvey

University of British Columbia

Engineering Physics 253

# Abstract

Presented here is the design of a novel two wheeled robot capable of competing in the 2013 ENPH 253 robot competition. The robot is designed to collect squash balls and shoot them at targets accurately and quickly. In accordance with the competition rules, the robot is designed to fit inside a 300mm3 cube and operate autonomously for at least 90 seconds. The robot will be able to follow tape at speeds up to 0.8m/s and shoot squash balls at 3-6m/s.

A lightweight spinning brush sweeps balls into a holding ramp for collection. The robot has two rear wheels that can be independently steered by two servo motors and has two ball casters mounted at the front. This wheel configuration allows the robot to navigate with differential steering while also being able strafe horizontally against a wall. Several reflectance sensors are utilized to track the location of black tape which is used for navigation.

The robot is controlled by the ubiquitous ATMega128 based Wiring board using code written in the Wiring language and Wiring IDE.

# Software Code and Algorithm

## Overview

The robot controller is the ATMega128 based Wiring board with a TINAH shield. The Wiring board runs programs created using the Wiring language and Wiring IDE.

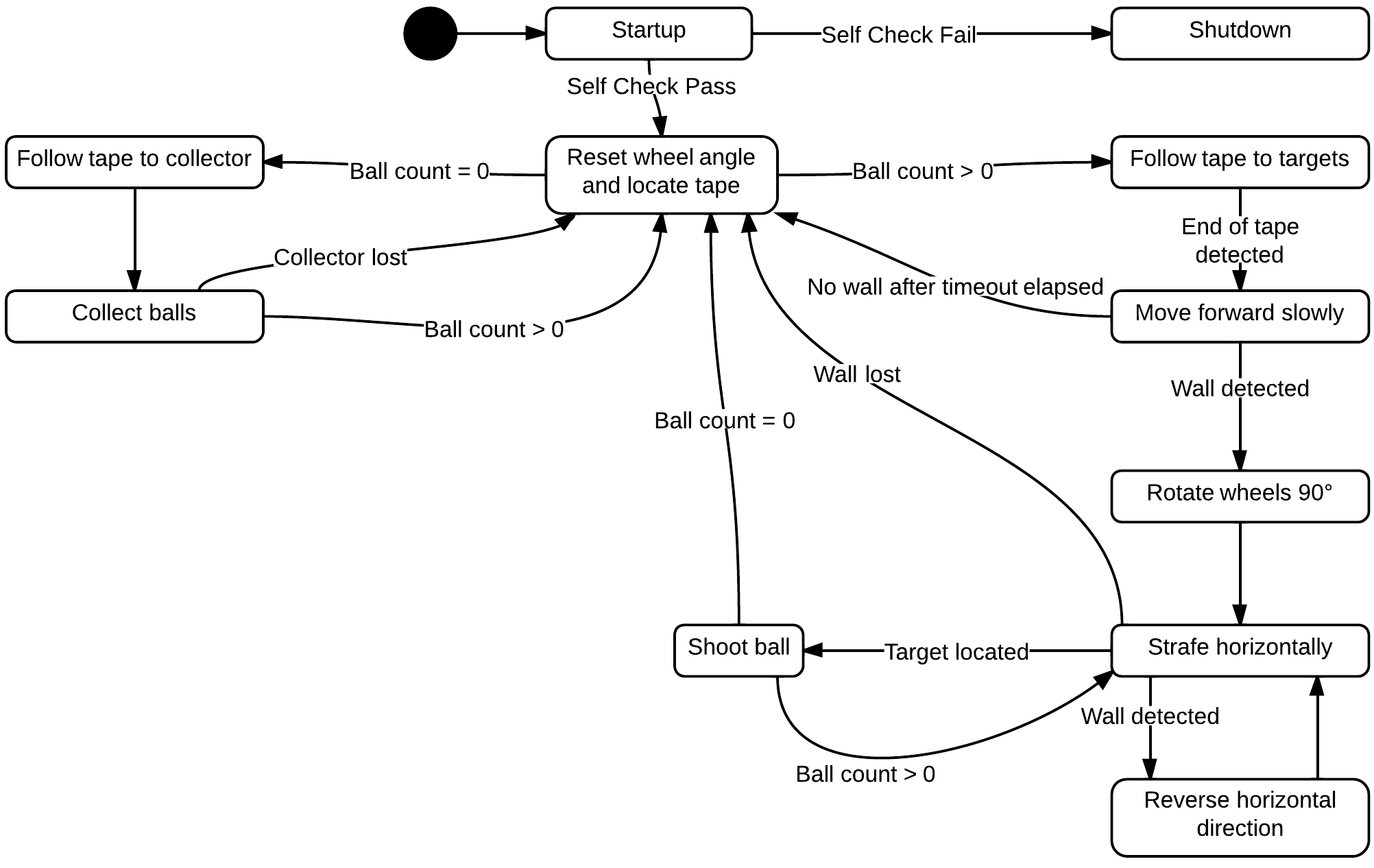
The robot can be modelled as a finite state machine. The highest level states are tape following, collecting, wall strafing, and shooting. The tape following state is central since it facilitates transitions between the collecting and wall strafing states. If the robot is disoriented at any point, this state provides moderate failure recovery since the robot can realign with the nearest tape. Shown below is the robot’s high level state diagram. 

Figure 1. Robot control is implemented as finite state machine.

## Locating Tape

The QRD sensor mounted on the tail permits tape detection while strafing. By counting the number of tape detections, it is possible to locate the centre tape. The below pseudocode provides one possible implementation of the centre tape algorithm. Since QRD readings are sampled dozens of times per second, it is important to note that this algorithm only counts *changes* in tape state. That is, it only detects tape when the previous reading did not detect tape. This ensures that each strip of tape is only counted a single time.

|  |
| --- |
| // PRE: Start facing target wall  // POST: Stopped with rear QRD sensor on center tape  void LocateCenterTape()  {  Strafe(LEFT);  while(!LeftTouchSensor()) // Wait for left wall alignment  Update();  Strafe(RIGHT); // Start moving right  int tapeCount = 0;  while(tapeCount < 2) // Look for center tape  {  // Prevents same tape from being detected twice  if (TapeRiseDetected()) tapeCount++;  else Update();  }  StopMotors(); // Stop at center tape  } |

Once the tape has been located, the next step is to move the robot onto the tape. This will be accomplished by rotating the robot until the front mounted QRD sensor has detected the tape. The following pseudocode demonstrates this manoeuvre.

|  |
| --- |
| // PRE: Situated on tape  // POST: Situated on tape, but rotated 180 degrees  void RotateOnTape(int turnRate = 100, unsigned int timeout = 10000)  {  SetWheelAngle(0);  LeftMotor(turnRate);  RightMotor(-turnRate);  while(!FrontTapeDetected())  {  Update();  if(timeout <= 0) TapeRecovery();  else timeout--;  }  StopMotors();  } |

## Tape Recovery

If the tape is unexpectedly lost and can’t immediately be recovered, a tape recovery algorithm will be called. This algorithm will cause the robot to move in an expanding spiral. This ensures that the tape closest to the robot will be detected first. It is a recursive algorithm that will keep looking for tape *ad infinitum*. It is difficult to implement a reliable tape recovery algorithm due to lack of sensor input and the limited computing power of the ATMega128. Because of this, heuristic algorithms such as the below algorithm will be considered for implementation.

|  |
| --- |
| // PRE: Sensors cannot detect tape  // POST: Moves in expanding spirals until tape is detected  void TapeRecovery()  {  int leftSpeed = -200;  int rightSpeed = 200;  int timeout = 0;  LeftMotor(leftSpeed);  RightMotor(rightSpeed);  while(!FrontTapeDetected())  {  Update();  timeout++;  // Increase spiral radius  if (timeout % 100 == 0) LeftMotor(++leftSpeed);  if (timeout < 10000) continue;  // Call again to reset spiral  TapeRecovery();  return;  }  StopMotors();  } |

## Tape Following

The robot has two tape following states, one that navigates to the collector (10 kHz emitter) and one that navigates towards the targets. A standard PID algorithm, as shown below, will be implemented to follow the tape. The end of the tape on the collector side does not have a tape intersection, while the target side does. The tape’s end will be detected by polling the two front-side mounted QRD sensors that act as intersection detectors.

// Computes new PID values and calculates a new motor speed for

// the left and right motors.

void Move()

{

if (leftDetected && rightDetected) error = 0;

else if (!leftDetected && rightDetected) error = TOO\_LEFT;

else if (leftDetected && !rightDetected) error = TOO\_RIGHT;

else if (!leftDetected && !rightDetected)

error = (previousError <= TOO\_LEFT) ? -OFF\_TAPE : OFF\_TAPE;

float proportional = (float)error \* proportionalGain;

float derivative = (float)(error - previousError) \* derivativeGain;

motor.speed(LEFT\_MOTOR, speed + (proportional + derivative));

motor.speed(RIGHT\_MOTOR, speed - (proportional + derivative));

previousError = error;

}

## Ball Collection

The ball collection manoeuvre is executed by driving the robot forward in the direction of the collector (10 kHz emitter). QRD reflectance sensors paired with comparators will be mounted in the ball hopper to detect successful ball collections. When a ball rolls past a QRD sensor, it causes a temporary state change from LOW to HIGH. This will trigger a standard hardware CPU interrupt that will increment the ball count.

## Wall Following

Two front mounted contact switches indicate whether the robot is currently contacting a wall. After the robot reaches the end of the tape on the target side, it will slowly approach the wall until contact is made. When this occurs, it will rotate the rear wheels by 90°. This wheel rotation lets the robot strafe horizontally along the wall. A PID algorithm will be implemented to keep the robot oriented aligned along the wall at all times.

## Target Acquisition and Shooting

Once a ball has been collected and the robot is aligned along the target wall, it will be considered capable of firing at the targets. Infrared sensors provide analogue input to the Wiring board to indicate whether the target has already been hit. If the input voltage is above a certain threshold then a valid target has been acquired. A PWM command will tell a servo to load the ball into the spinning disks and fire.

Decision theory influences the shooting behaviour. If no targets can be detected, then either the robot’s sensors are not functioning properly or all of the targets have been hit. It is assumed that the robot is more likely to fail than it is to succeed. With this assumption, it is in the robot’s best interest to blindly fire at targets. If all targets had been hit, it implies that the robot is doing very well, thus making the deleterious effects of unnecessary firing negligible. The below flow chart provides an overview of the shooting algorithm. The *wall count* variable is used to determine whether no targets were detected.

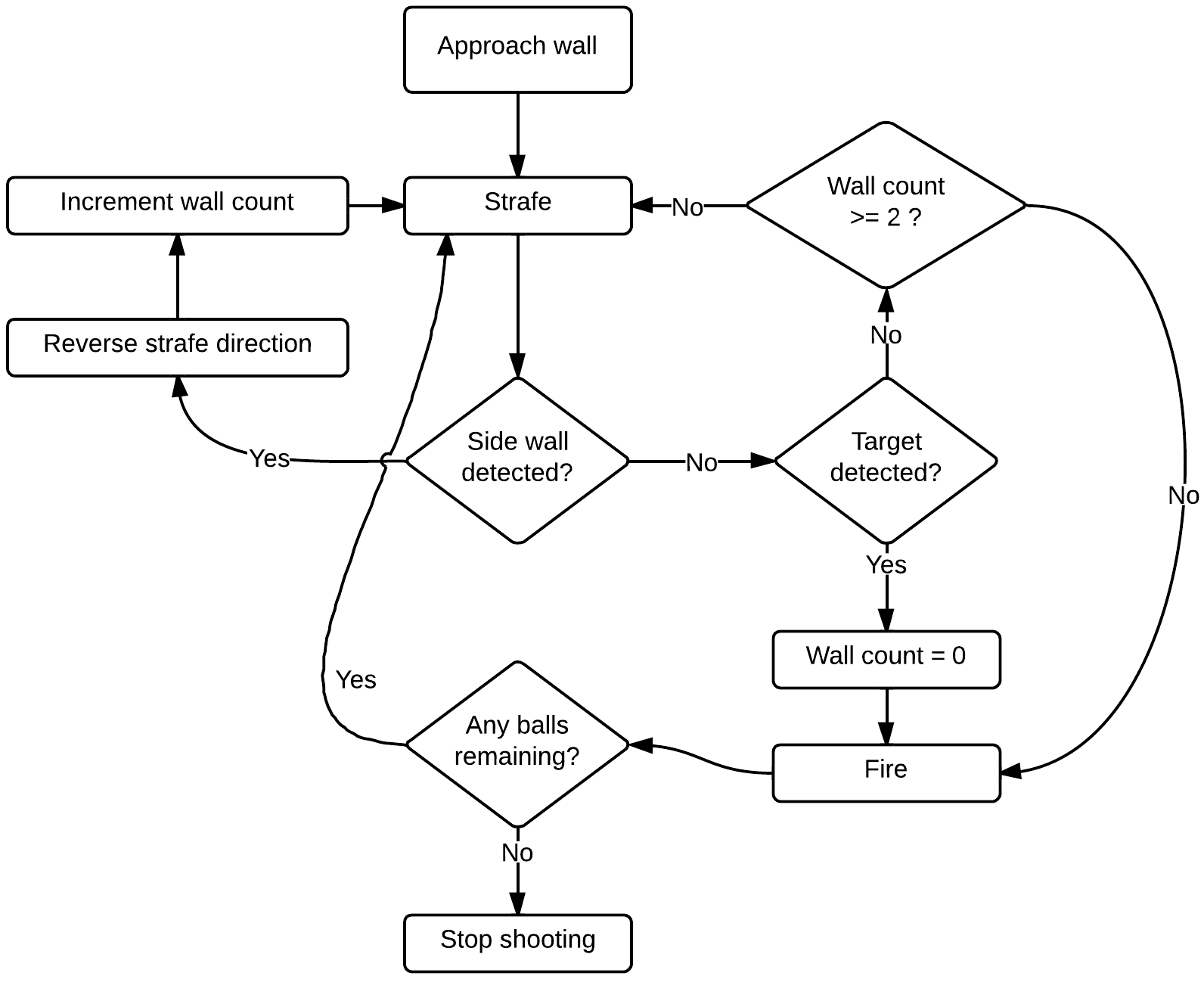


Figure 2. The shooting algorithm flowchart. When no balls remain, the robot stops shooting and enters a ball collection state.