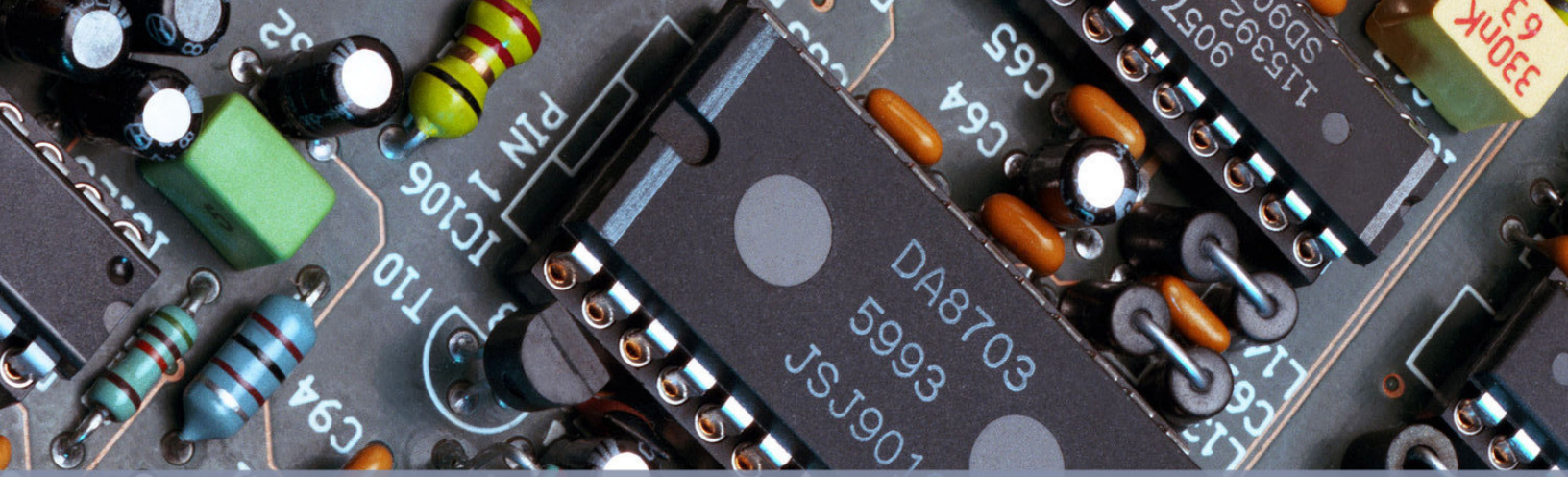
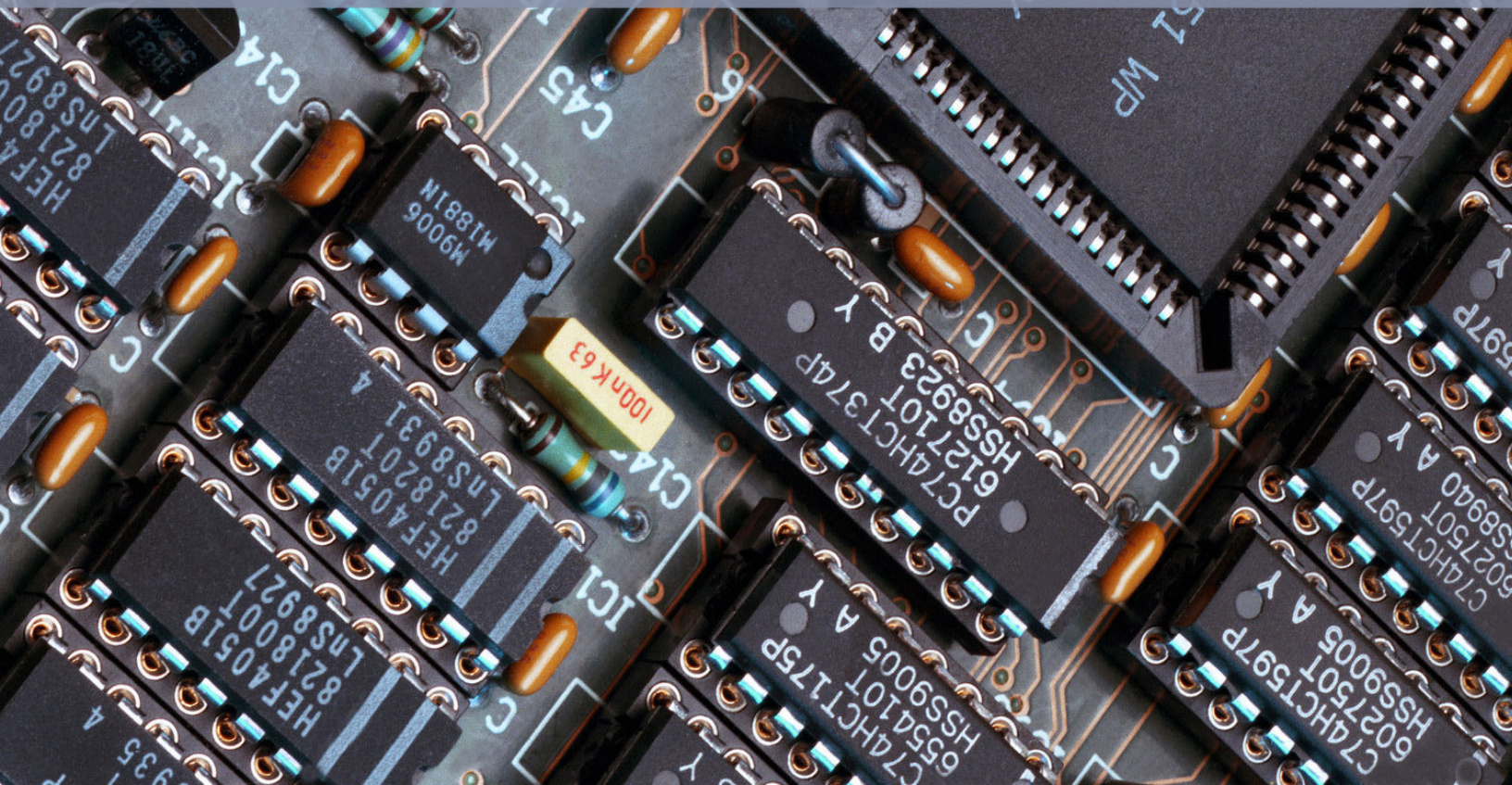


Battle Bot

MAE 3780 Final Report

Bin 42: Sarah Angle, Maxwell Dergosits, and Natalie Moore



1 Introduction

When we started looking at designs for our sumo-bot, we prioritized a few components. First was speed of detection. The engaging robot in a match usually has an advantage if it can start acting offensively first. Also, we wanted to emphasize pushing power, as this is the way that most matches are won. To do both these things, we focused on our robot's H-Bridge, motors, and wheels. Once these were organized, we built the robot's body around those components. Over the course of this report, we will detail the conceptualization and build of our robot.

2 Design

Overall, our design uses the basic battle bot chassis and mounts provided. Our robot is two wheel drive, and incorporates 2 sonar and 5 QTI sensors. Additionally, we have a third continuous rotation servo mounted on the top of our robot which acts as a flipping mechanism by actuating a hinged plate.

2.1 Mechanical

The obviously unique portion of our robot is its plastic 'armor.' A sheet of .125" acrylic was laser cut to create a set of plates. These were then hinged to a top plate, held above the chassis by a screw standoff. The plates were designed such that they stood at about a 15 degree angle from vertical. The motivation behind this design was twofold. The plates act defensively, as someone trying to flip the robot from under one of these plates will instead flip the plate, leaving our robot relatively unharmed. Also, the front plate is actuated to act as an offensive flipper. The plates have cutouts which avoid interfering with the side wheels and front rollerball. Each plate also has cutouts to allow for sensor placement, sonar sensors in front and IR sensors on sides. The top plate has a cutout that allows the servo to be seated and screwed in place.
Additionally, we purchased from Parallax slightly wider and more frictional wheels than the original ones provided. This should allow our robot to push harder and resist sliding. For pictures that show all of these elements, please see appendix Figure 2 and Figure 3.

2.2 Sensor Placement

We had three main sensors on our robot. The first, our QTI or line sensors, were arranged under our chassis. The center QTI was used to signal the end of battle. The other four, however, were used by our robot to sense when we needed to begin evading the match boundary. They are placed as far from the center of mass to allow the most reaction time for the robot. We used five of these sensors because it gave us warning in all four directions, such that each sensor's warning had a distinct reaction. There is no direction the robot can be pushed out of the ring that does not trigger one of these warnings. Please see Figure 7 in appendix for QTI placement.

The second type of sensors were sonar detectors. The two of these were mounted on the front plate of the robot. We chose to use two sonars because it gives us directional detection, i.e. our robot knows the opponent's direction in case it loses the opponent, allowing it to swivel in that direction instead of restarting the search. We originally wanted the sonar sensors to be angled slightly outward, but found that mounting to be unstable.

The final sensor on our robot was a internal use touch sensor. In order to lift our flipper gate, we had to be able to put it back down in the proper position, or our sonar would be blind and the next lift would not be timed correctly. Thus, a whisker was placed so that it touched a small metal electrode on the front plate when the plate was in the down position, but not in the lifted position. When this switch activated, it told the robot that the plate had been lowered enough. Please see appendix Figure 4 for detail.

We decided against using IR sensors because their range was very small, measured as less than an inch. Additionally, their measurements were inconsistent and unreliable.

2.3 H-Bridge

When creating our H-Bridges (one for each motor,) we first had to decided what premium we were willing to put on its performance. That is, since the H-Bridge's effectiveness is highly dependent on the quality (and thus, price) of the transistors used, we needed to find a budget. We chose approximately 10 dollars based on the other parts we already deemed critical to our design. For more of our budget, please see appendix Figure 6 for the Bill of Materials. We then chose a schematic based on some online research. We chose our schematic because it was simple, effective (as claimed by reviewers), and took advantage of MOSFETs over BJTs. Please see Figure 8 in appendix for a circuit diagram of our H-Bridge.

From there, we picked a set of N-type and P-type MOSFETS to use. We determined the quality of the FETs using two characteristics, maximum current and 'stolen' voltage. The formula for voltage drop across a transistor is:

$$V = R_{ds} * I_d \tag{1}$$

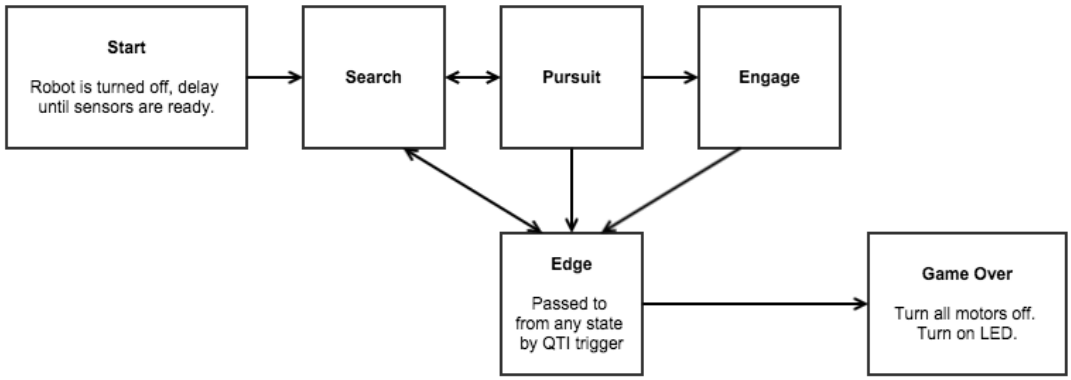
Based on these criteria, we chose two approximately equal quality FETs. Our N-Chans, costing $0.66 apiece, carry up to 5A, and have a maximum .25V voltage drop. The P-Chans, costing $1.01 each, carry up to 8.6A and have a maximum .75V voltage drop. P-Chan MOSFETs are significantly more expensive for the same voltage drop as N-Chan MOSFETs.

We used a third H-Bridge to drive the lifting motor, but this was a lower priority. For this reason, we used the simple BJT H-Bridge we implemented in an earlier lab using provided components. See appendix Figure 5 for our H-Bridge breadboard.

3 Strategy

Our robot's strategy was rather simple. We designed it to be a state machine, with five states which complete certain actions and pass to each other under certain conditions.

Figure 1: The between state (overall) flow chart.



In the appendix we have included flow charts for each of the four main states: search, pursue, engage, evade (edge). These were translated into C functions for the microprocessor to perform. However, doing this required a fair amount of ingenuity so that it was executed as we imagined. For this reason, we highlight some of these innovations in the next section.

4 Software Implementation

In this section, we highlight some key developments in our C code that were non-trivial implementations.

We organized our code into an explicit state machine where states were functions and arguments and return values were the state transitions. We thought this part of the code was interesting because using switch statements made the code very clean and readable. Because of this each state and action was clearly defined and not dependent on the other states.

```
while(1) {
  switch (dir) {
    case FOUND:
      dir = pursue();
      break;
    case QTI_TRIGGERED:
      dir = evade();
      break;
    case ENGAGE:
      dir = engage();
      break;
    case SEARCH_LEFT:
    case SEARCH_RIGHT:
```

```
      dir = search(dir);
      break;
    default:
      dir = SEARCH_LEFT;
      break;
  };
}
```

We made sure that we didnt have very many magic numbers throughout our code and whenever we used a numerical constant value we made sure it was in an enum rather than repeating the exact value all over the code.

```
enum delays {
  EVADE_TURN_DELAY=300,
  EVADE_STRAIGHT_DELAY=300,
  LIFT_DELAY = 900,
};
// distances for the sonar
enum distances {
  DETECT_DISTANCE=40,
  ENGAGE_DISTANCE=4,
};
```

To make sure that we wouldnt miss QTI interrupts during a delay we only delayed for 10ms at a time and checked that no QTIs had tripped. When testing we noticed that we would go over the line during delays because there can be a lot of movement in 300 milliseconds, so checking and responding to the QTIs every 10ms alleviated that problem.

```
int time_left = EVADE_TURN_DELAY;
 while(time_left > 0) {
   setdelay(10);
   waitfordelay();
   time_left-=10;
   if (QTI_TRIPPED) {
     return EVADE;
   }
 }
```

To make sure that we instantly stopped what we were doing if the center QTI tripped, we had an interrupt call the following function, which never returned. This way we didnt need to write this code every time we checked the QTI.

```
void end(char q) {
  if (CENTER_QTI) {
    turn_on_led();
    end_lift();
    move_motors(stop);
    while(1);
  }
}
```

During search we did nothing except for check both the sonars and QTIs. This and our well designed H-bridge made us find the other robot very fast. Searching in the direction it was last seen in helped us not have to spin all the way around in case we overshot our target.

```
enum action search(enum action d) {
  if (d==SEARCH_LEFT) {
    move_motors(sleft);
  } else {
    move_motors(sright);
  }

  while(1) {
    if (QTI_TRIPPED) {
      return QTI_TRIGGERED;
    }
    float distleft = left_sonar();
    float distright = right_sonar();
    if ((distleft<DETECT_DISTANCE) && (distright<DETECT_DISTANCE)){
      return FOUND;
    }
  }
}
```

Finally, rather than using the built in delay micro and millisecond function we used timer1 to do delays which is much more accurate. We also were able to use the time during these delays to perform other functions, rather than wasting CPU cycles.

```
void setdelay(uint16_t time) {
  DISABLE_INT
    time1 = time;
  ENABLE_INT
}

void waitfordelay(void) {
  while(1) {
    char done = 0;
    DISABLE_INT
```

```
      if (time1 == 0) {
        done =1;
      }
    ENABLE_INT
      if(done) {
        return;
      }
  }
}
```

5 Competition Performance

During the competition, our robot won 3 matches and lost 2. Our robot found the other robot very quickly each time, and never drove itself out of bounds. The lifting mechanism wasnt a very effective active element because most of the opposing robots were too heavy to be affected by it. However, our motors were very powerful compared to the other robots and we won whenever the match came down to a pushing contest.

We lost two matches because we were pushed over before we could push the other robot out of bounds. Our robot was narrower side to side than the front to back, so when another robot tried to flip our robot from the side, it wasnt stable enough to stay upright. Our acrylic defense mechanism was designed so that other robots would only flip up the plastic piece and not the entire robot, but the two robots we lost against got their flipping mechanism under our wheel. If we could repeat this project, we would make our robot lower to the ground so it would be more stable, and also add a more effective flipping mechanism that would get underneath other robots and flip them over.

6 Evaluation

One of the robots biggest strengths was the H-bridge design that used both P-channel and N-channel MOSFETs. The original H-bridge design used in lab used BJT transistors that could only supply a maximum of 500 mA of current to the motor, but our design used two FQPF11P06 P-channel MOSFETs that could output a max of 8.6 A each and four N-channel 2SK4017 MOSFETs that could output a max of 5 A each. Because the voltage supply to the H-bridge was 6V and the maximum gate-source voltage for the P-channel MOSFETs was 25 V, we added 1000 resistors across the gate and source pins of the MOSFET. This improved the speed and power of our motor because the voltage across the ground and source of the MOSFET was limited. Overall, because more current could be supplied to the motor, the motor turned at a faster speed and with more torque than when the BJT H-bridge was used.

Another major strength of our robot was the protective acrylic shell that surrounded it. One of the most common active elements used by other teams was a flipper that would get under the other robots and flip them over. To prevent our robot from being flipped, we

attached pieces of acrylic on each side with hinges, so that when a team tried to flip the robot over, they would only flip the acrylic up and not the entire robot.

Along with the mechanical and electrical design, there were several strengths in our coding. We formatted our code into a state machine with four different states: pursue, evade, engage, and search, which were the four tasks that our robot needed to perform throughout the rounds of competition. Because we divided our code into a state machine, it was easy to switch from one task to another, and prevent the robot from going out of bounds while enabling it to search and push out the other robot.

Our robot had several weaknesses with the overall design. Our lifting mechanism's motor was controlled by a BJT H-Bridge. Because there wasnt room in our budget to power it with a MOSFET H-bridge, there wasnt enough power supplied to the motor to lift it quickly or with a lot of force. This caused our active element to not be very effective at flipping other robots over, so we had to rely on push strength to knock the other robot out of the ring. That said, our pushing mechanism was quite strong thanks to the H-Bridge described above. Another weakness we had was our robot's stability. The width was 8 cm, the length was 13 cm, and the height was around 16 cm. The small width allowed it to be easily knocked over if it was pushed or flipped from the side. If it had been lower to the ground, or had the robot been made wider, it would have been more stable and less likely to fall over when hit by another robot.

7 Statement of Work

Each member in this group had an equal share of work. Additionally, each member had a hand in each process, that is, any member of this group could detail a majority of any of the sections above. That said, Sarah focused mostly on the mechanical work, like assembly, hardware integration, and physical design. Natalie focused on the H-Bridge and hardware implementation. Maxwell focused on software implementation.

8 Conclusion

Our robot performed fairly well in the competition. It was very good at finding the opponent, but had some problems executing an offensive strategy. We learned that the H-Bridge was a very important component of our robot, and we were pleased that we invested time and budget into making ours efficient. We seemed to move faster and have greater pushing power than most robots we challenged. Additionally, our robot performed dependably; at no point in a match did a sensor, motor, etc. fail. We competed well, and with a few minor tweaks, this robot could have performed even better.

Appendix

Robot Pictures

Figure 2: Robot in Profile.

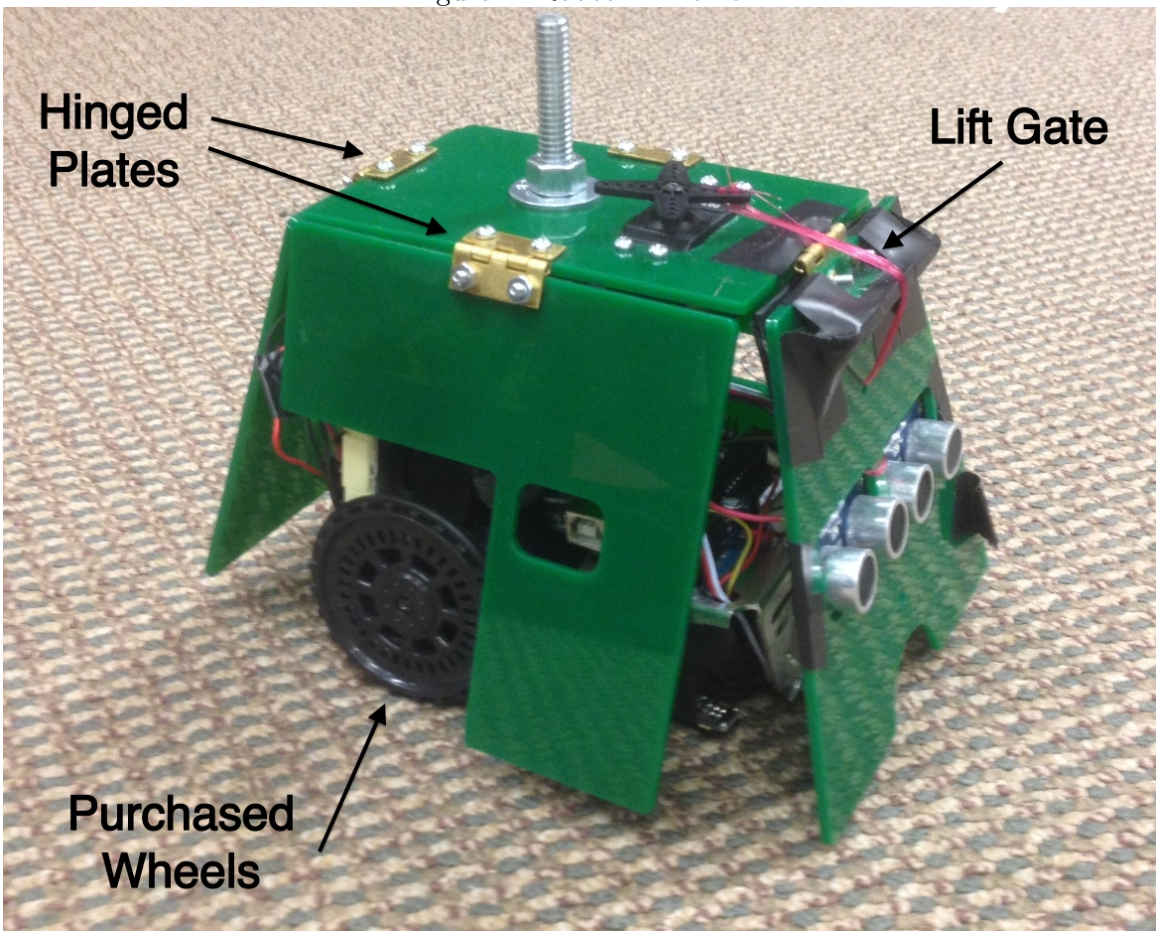


Figure 3: Robot from Front.

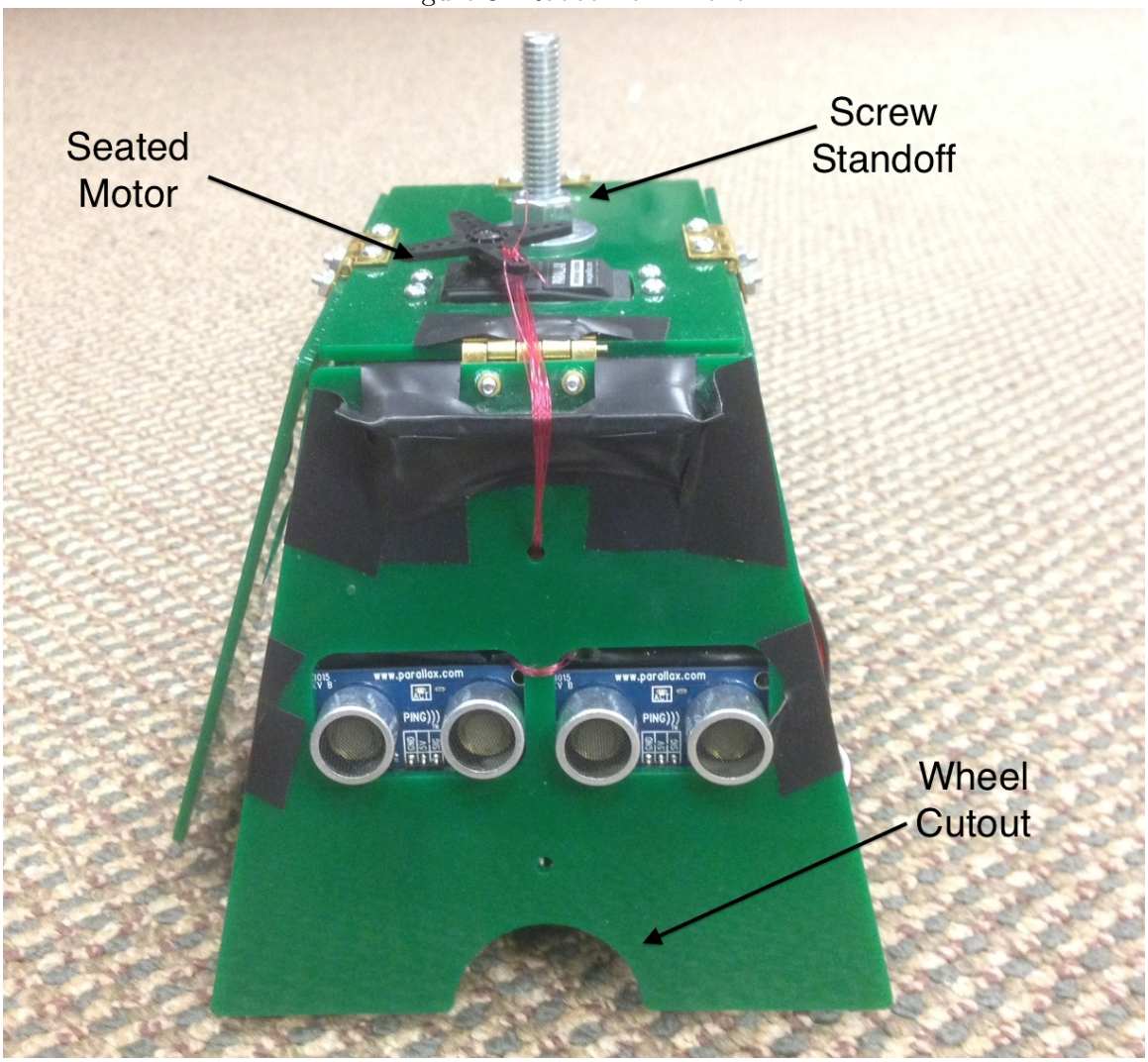


Figure 4: Lifter Switch Detail.

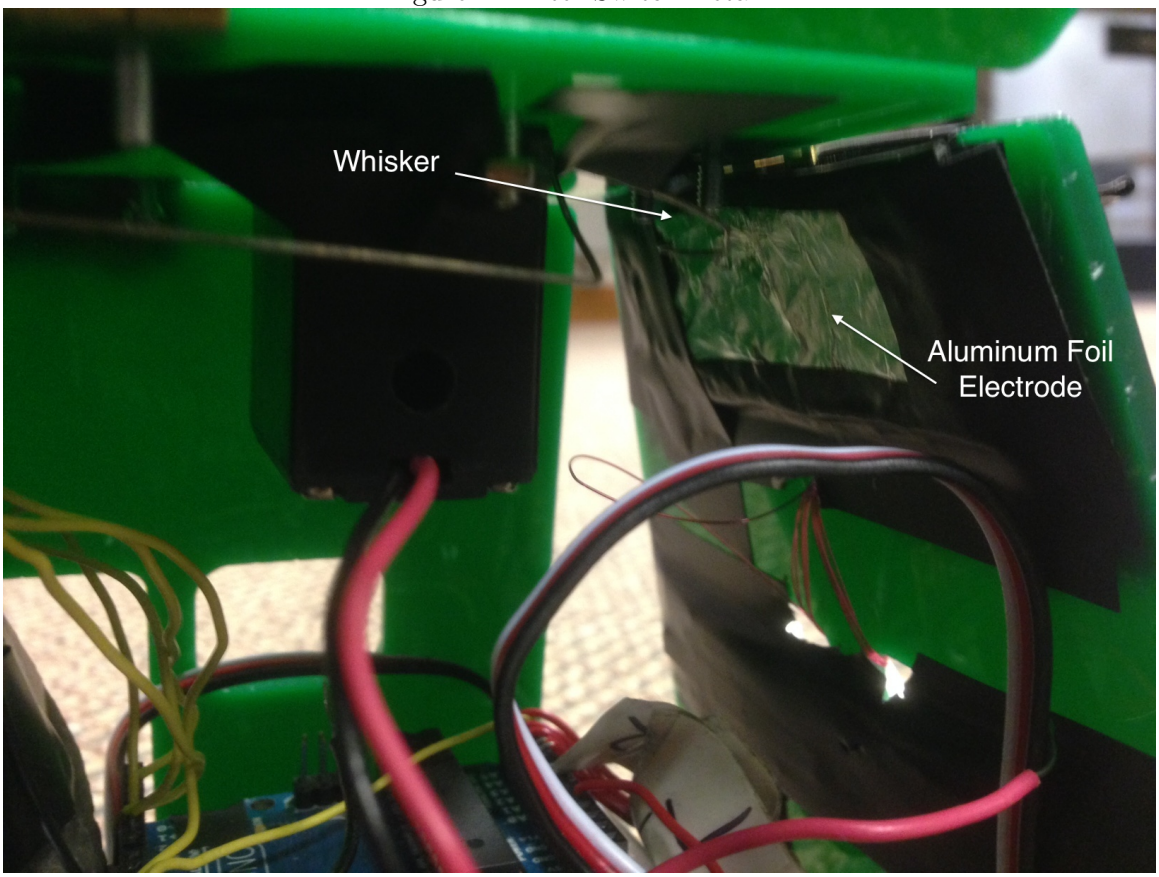
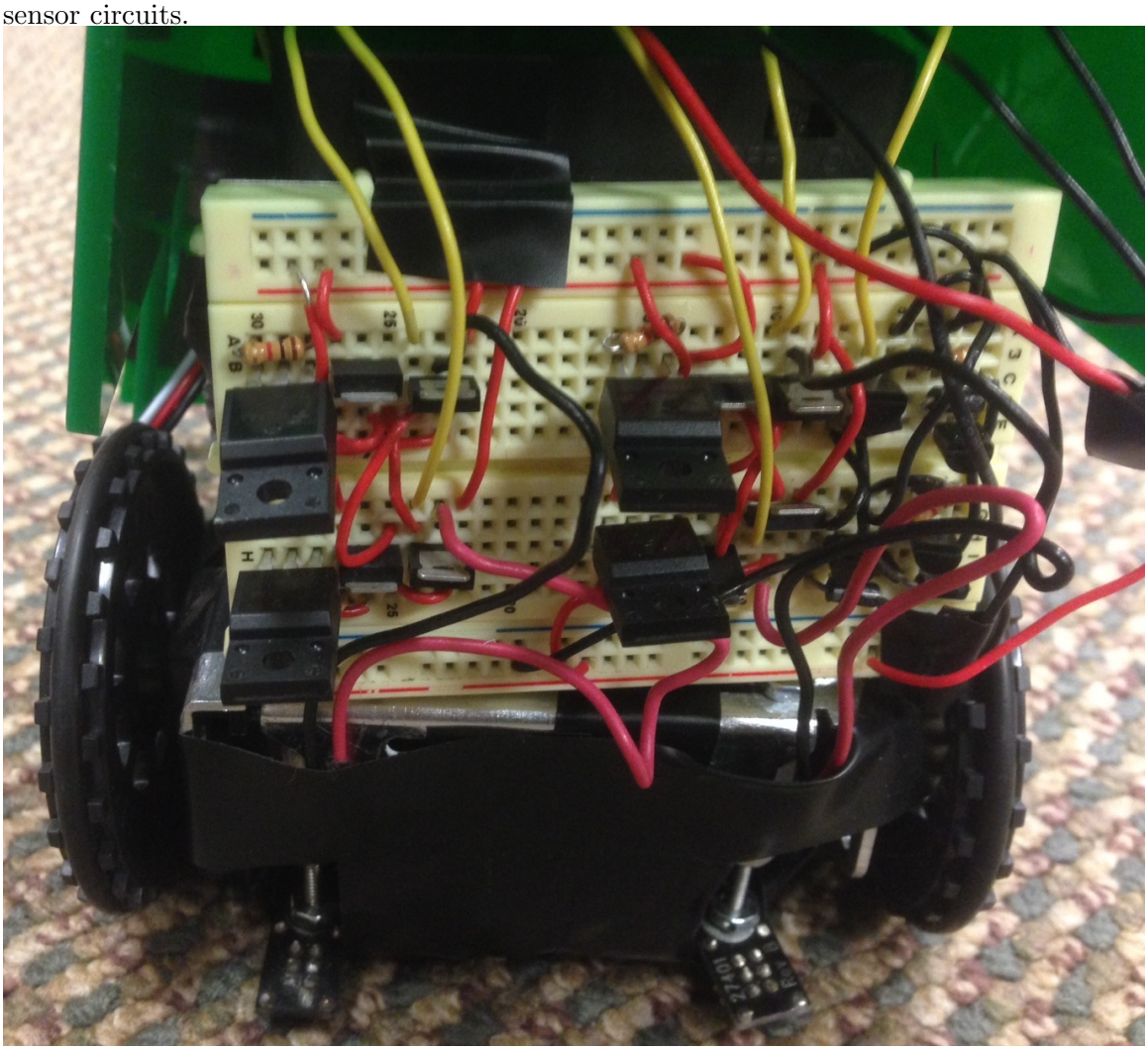


Figure 5: H-Bridge Breadboard. Note that this is the only part of our circuitry on a breadboard. This was very bulky, and soldered circuit boards were far more effective for sensor circuits.



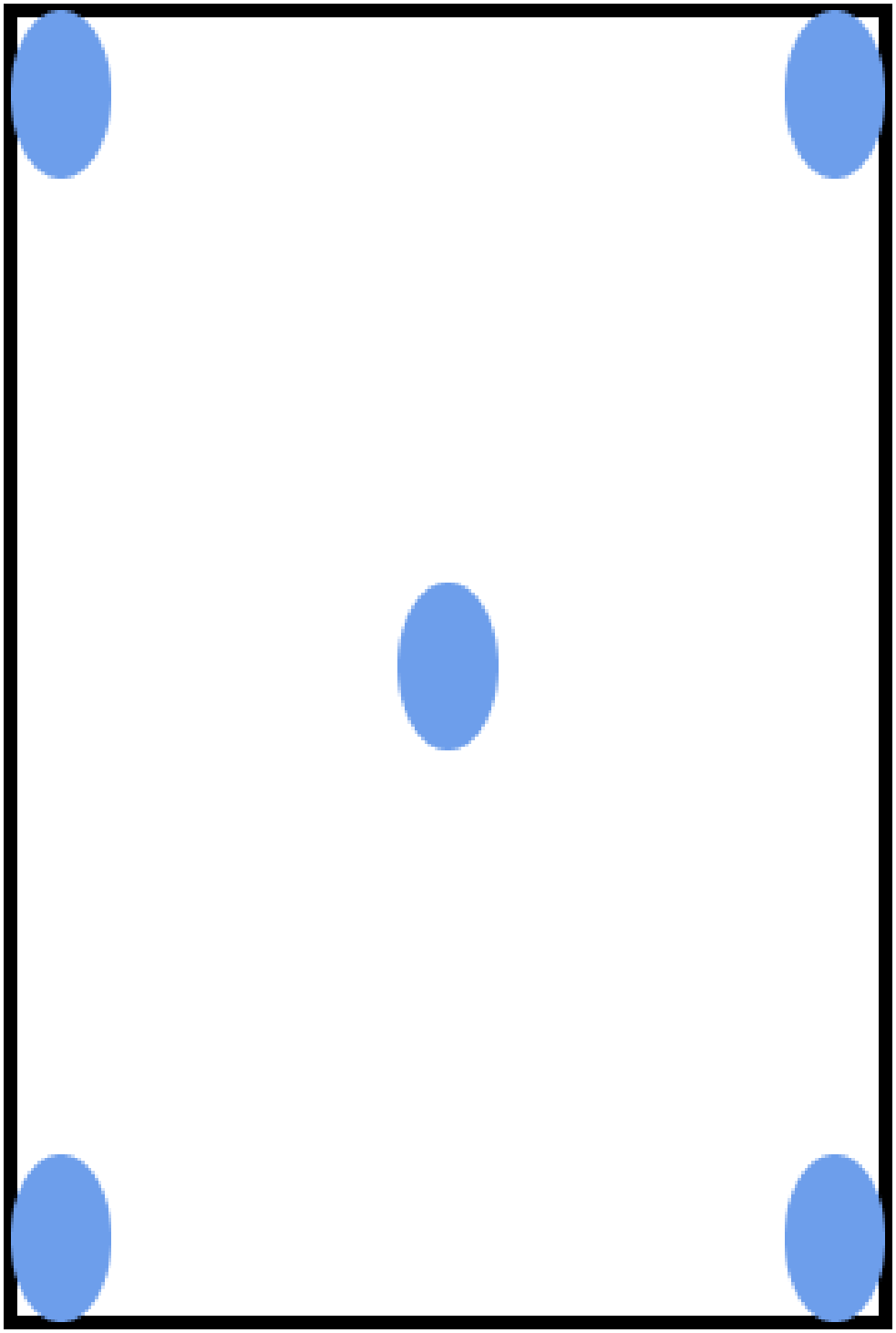
Budget of Materials

Figure 6: Project Budget of Materials. According to project details, the budget must be under $40.00.

Line #	Part	Source	Part Number	Quantity	Cost
1	Acrylic - 12” x 12” Sheet - Colored	McMaster	8505K11	1	4.50
2	Thermoplastic Inserts	McMaster	93365A110	1 (Bag of 100)	1.00
3	Hinges	McMaster	1603A3	4	6.44
4	Wheels	Parallax		2	7.00
6	Sonar	3780 Lab	N/A	1	5.00
7	Continusous Rotation Servo	3780 Lab	N/A	1	3.00
8	QTI Sensors	3780 Lab	N/A	2	4.00
9	H-Bridge N-Chans	Had on hand	Have	6	3.96
10	H-Bridge P-Chans	DigiKey	FQPF11P06FS-ND	4	4.04
					38.94

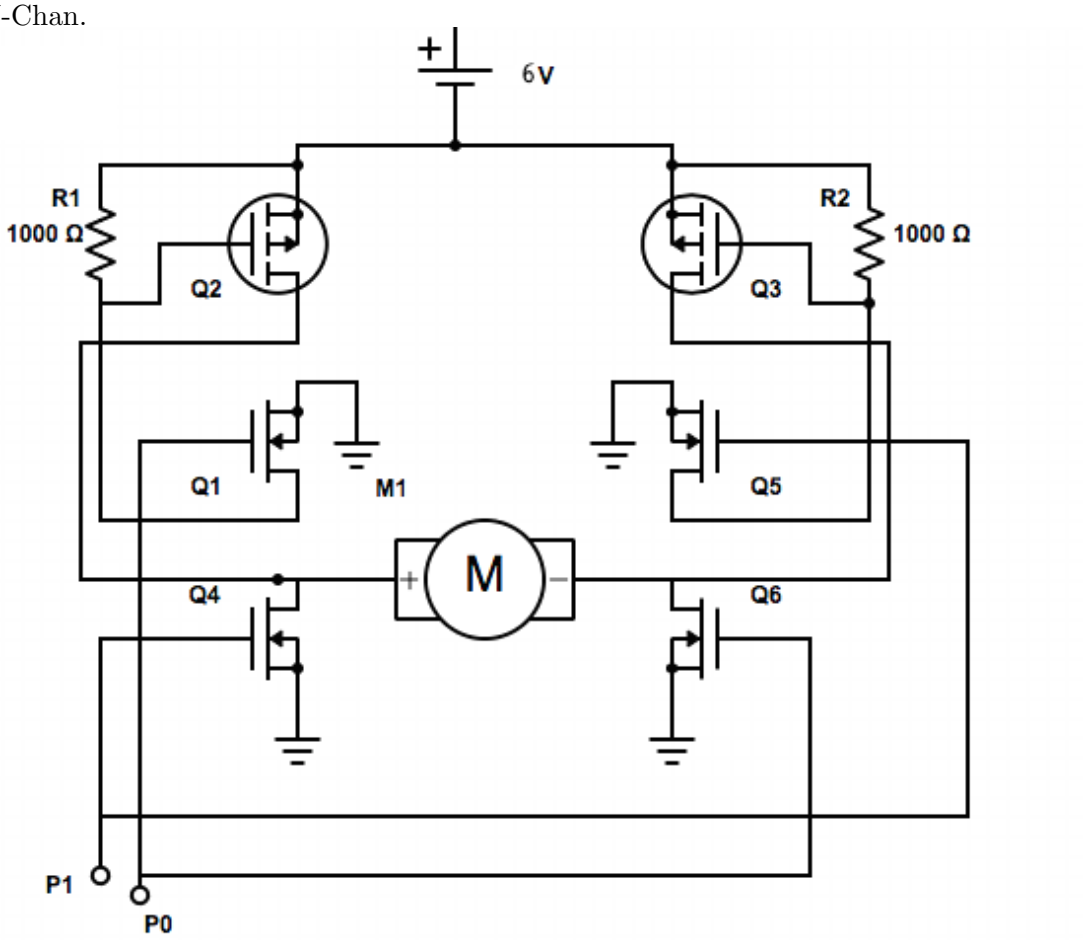
Sensors

Figure 7: QTI Placement



H-Bridge

Figure 8: H-Bridge Circuit. Note that Q2 and Q3 are P-Chan MOSFETs, while all others are N-Chan.



State Flow Charts

Figure 9: The Search state flow chart.

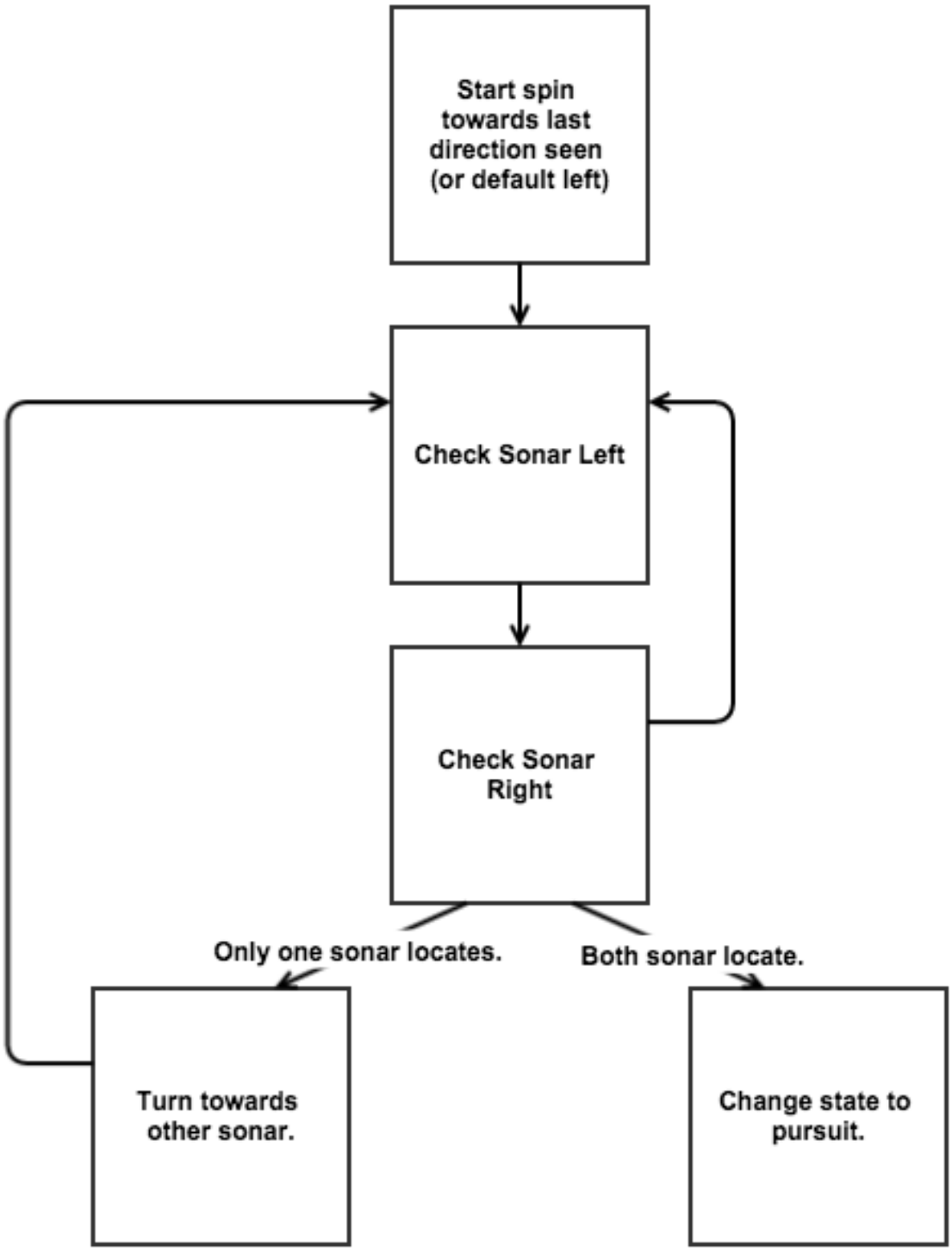


Figure 10: The Pursue state flow chart.

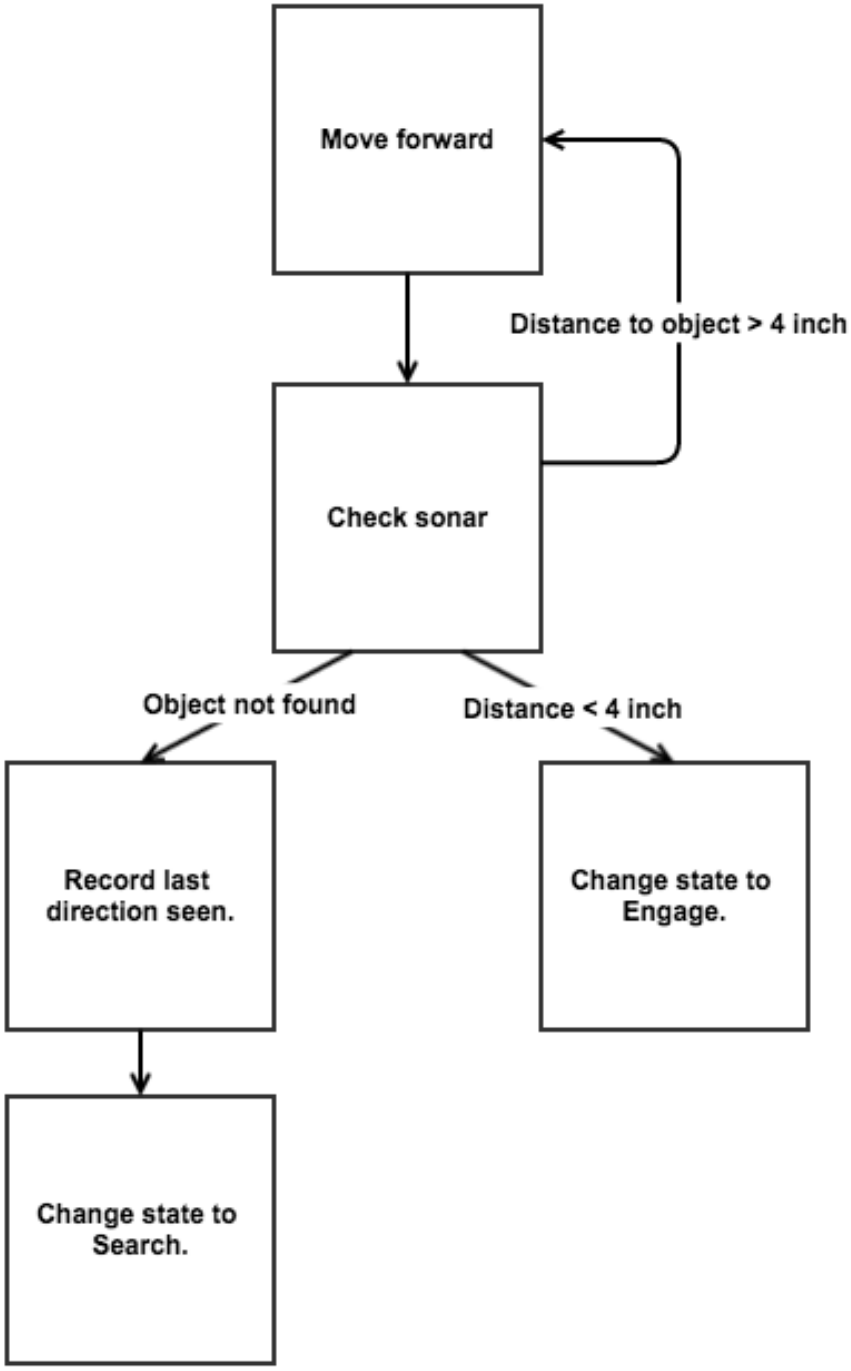


Figure 11: The Engage state flow chart.

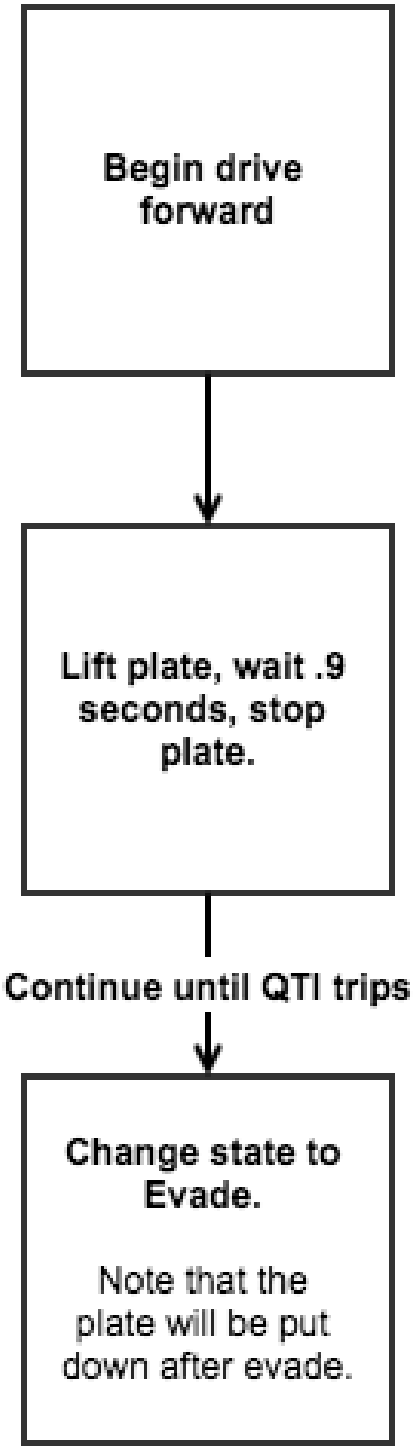
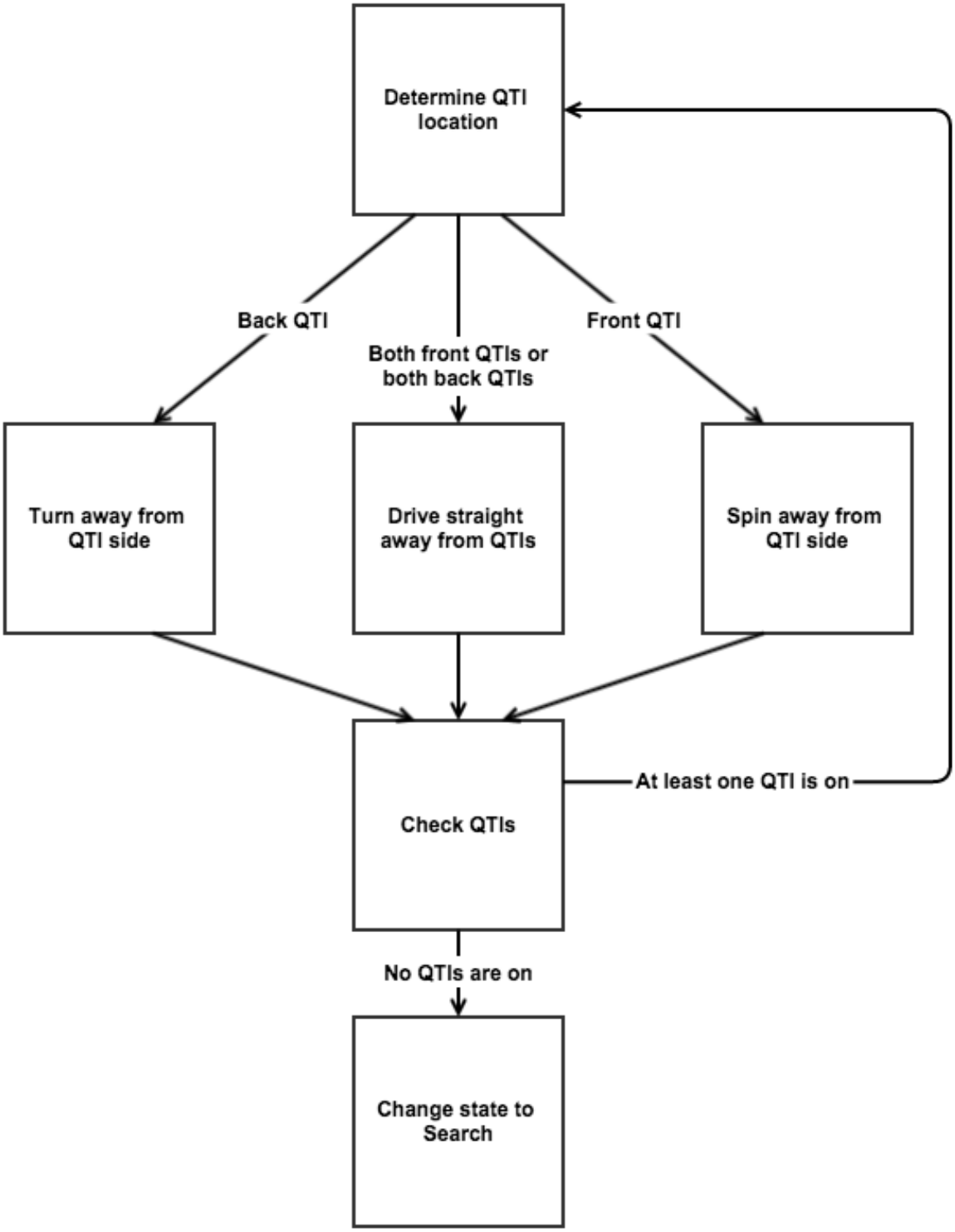


Figure 12: The Evade state flow chart.



Program Listing

```
//timers.c

#include "timers.h"

int16_t time1 = 0;
ISR(TIMER0_COMPA_vect) {

  DISABLE_INT
    if (time1>0)  --time1;
  ENABLE_INT

}
void init_timer0(void) {

  //set up timer 0 for 1 mSec timebase
  TIMSK0= (1<<OCIE0A);  //turn on timer 0 cmp match ISR
  OCR0A = 249;      //set the compare register to 250 time ticks
  //set prescalar to divide by 64
  TCCR0B= 3;
  // turn on clear-on-match
  TCCR0A= (1<<WGM01) ;
}

void setdelay(uint16_t time) {
  DISABLE_INT
    time1 = time;
  ENABLE_INT
}

void waitfordelay(void) {
  while(1) {
    char done = 0;
    DISABLE_INT
      if (time1 == 0) {
        done =1;
      }
    ENABLE_INT
      if(done) {
        return;
      }
  }
}
```

```
//timers.h

#ifndef _TIMERS_H_
#define _TIMERS_H_
#include <avr/io.h>
#include <avr/interrupt.h>
#define DISABLE_INT char sreg; sreg = SREG; cli();
#define ENABLE_INT SREG = sreg;
```

```
void init_timer0(void);
void setdelay(uint16_t time);
void waitfordelay(void);
#endif
```

```
//sonar.c

#include "motor.h"
#include "qti.h"
#include "sonar.h"
#include "timers.h"
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
// Sonar is on pins 12(PB4) and 13(PB5)
// PCINT4 PCINT5
float distance = 10000;

char left_on = 0;

char right_on = 0;
char x = 0;

void setTCNT1(uint16_t value) {
  unsigned char sreg;
  sreg = SREG;
  cli();
  TCNT1 = value;
  SREG = sreg;
}

uint16_t getTCNT1(void){
  uint16_t value;
  unsigned char sreg;
  sreg = SREG;
  cli();
  value = TCNT1;
  SREG = sreg;
  return value;
}

ISR(PCINT0_vect){
  if (x==0){
    setTCNT1(0);
    x=1;
  }else{
    x=0;
    PCMSK0 &= ~(1<<PCINT4);
    PCMSK0 &= ~(1<<PCINT5);
    int16_t distance_count = getTCNT1();

    // refer to prelab 7 for the explicit calculation
```

```
    distance = ((float)distance_count) * 0.02712;
  }
}

float left_sonar() {
  distance=100;

  DDRB |= (1<<PB4);
  PORTB |= (1<<PB4);

  _delay_us(5);

  DDRB &= ~(1<<PB4);
  PORTB &= ~(1<<PB4);
  left_on = 1;
  right_on = 0;
  PCICR |= (1<<PCIE0);
  PCMSK0 |= (1<<PCINT4);
  PCMSK0 &= ~(1<<PCINT5);
  x= 0;
  _delay_ms(20);
  return distance;
}

float right_sonar() {
  distance=100;

  DDRB |= (1<<PB5);
  PORTB |= (1<<PB5);

  _delay_us(5);

  DDRB &= ~(1<<PB5);
  PORTB &= ~(1<<PB5);
  left_on = 1;
  right_on = 0;
  PCICR |= (1<<PCIE0);
  PCMSK0 |= (1<<PCINT5);
  PCMSK0 &= ~(1<<PCINT4);
  x= 0;
  _delay_ms(20);
  return distance;
}

void init_sonar(void) {
  // turn on the right prescalar
  TCCR1B |= (1<<CS11) | (1<<CS10);
  TCCR1B &= ~(1<<CS12);
}
```

```
//sonar.h

#ifndef _SONAR_H_
```

```
#define _SONAR_H_
float left_sonar();
float right_sonar();
void init_sonar(void);
#endif
```

```
//qti.c

#include <avr/interrupt.h>
#include <avr/io.h>

#include "qti.h"
void (*stop)(char);

ISR(PCINT1_vect) {

  qti = ~PINC;
  qti &= 0b00011111;
  stop(qti);
  //if (CENTER_QTI) {stop(); }
}

void init_qti(void (*f)(char)) {

  stop = f;
  DDRC &= ~((1<<PC0) | (1<<PC1) | (1<<PC2) | (1<<PC3) | (1<<PC4)); // Make
    sure that the pins are inputs
  PORTC &= ~((1<<PC0) | (1<<PC1) | (1<<PC2) | (1<<PC3) | (1<<PC4)); // Make
    sure that the pullups are turned off

  PCICR |= (1<<PCIE1); // enable PCINT1_vect (group)

  PCMSK1 |= ((1<<PCINT8) | (1<<PCINT9) | (1<<PCINT10) | (1<<PCINT11) | (1<<
    PCINT12)); // enable specific interrupts
  qti = 0;
}

void clear_sensors(void) {
  qti = 0;
}
```

```
//qti.h

#ifndef _QTI_H_
#define _QTI_H_
#define NUM_QTI (5)
char qti;

#define FRIGHT_QTI ((qti &(1<<3)))  //A3
#define FLEFT_QTI ((qti &(1<<1)))   //A4
```

```
#define BRIGHT_QTI ((qti &(1<<2)))  //A1
#define BLEFT_QTI ((qti & (1<<4)))   //A2
#define CENTER_QTI ((qti & (1<<0)))   //A0
#define QTI_TRIPPED (qti != 0)

#define WAIT_FOR_QTI_OFF(x) {while(x) {setdelay(20); waitfordelay(); }}
#define WAIT_FOR_QTI_ON(x) {while(!x) {setdelay(20); waitfordelay(); }}
void init_qti(void(*f)(char));
void clear_sensors(void);

#endif
```

```
//motor.c

#include "motor.h"

void init_motors(void)
{
  // set pins as outputs
  DDRD |= (1<<PD6);
  DDRD |= (1<<PD5);
  DDRD |= (1<<PD3);
  DDRB |= (1<<PB3);
}

void left_foward()
{
  PORTD &= ~(1<<PD5); // pin off
  PORTD |=  (1<<PD6); // pin on
}
void left_reverse()
{
  PORTD |=  (1<<PD5);
  PORTD &= ~(1<<PD6);
}

void left_stop()
{
  PORTD &= ~(1<<PD6);
  PORTD &= ~(1<<PD5);
}

void right_reverse()
{
  PORTB &= ~(1<<PB3);
  PORTD |=  (1<<PD3);
}
void right_stop()
{
  PORTB &=~(1<<PB3);
  PORTD &=~(1<<PD3);
}
void right_foward()
```

```
{
  PORTB |= (1<<PB3);
  PORTD &=~(1<<PD3);
}

void move_motors(enum dir direction) {
  switch(direction) {
    case mleft:
      left_stop();
      right_foward();
      break;
    case mright:
      right_stop();
      left_foward();
      break;
    case sleft:
      left_reverse();
      right_foward();
      break;
    case sright:
      left_foward();
      right_reverse();
      break;
    case straight:
      left_foward();
      right_foward();
      break;
    case reverse:
      left_reverse();
      right_reverse();
      break;
    case stop:
      left_stop();
      right_stop();
      break;
  };
}
```

```
//motor.h

#ifndef _MOTOR_H_
#define _MOTOR_H_
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

#include "qti.h"
#define LEFT_HIGH (255)
#define RIGHT_HIGH (255)
#define LEFT_LOW (1)
#define RIGHT_LOW (1)
#define SPIN_DELAY (580)
```

```
enum dir {
  mleft, // turn left wheel off move right wheel foward to turn left
  mright,// turn right wheel off move left wheel foward to turn left
  sleft, // spin left, left reverse, right forward
  sright, // spin right, right reverse, left forward
  straight,
  stop,
  reverse
};

void move_motors(enum dir direction);
void init_motors(void);
void left_foward();
void left_reverse();
void left_stop();
void right_foward();
void right_reverse();
void right_stop();
#endif
```

```
//Our main function

#include "motor.h"
#include "qti.h"
#include "sonar.h"
#include "timers.h"
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
// the different delays for evade and engage
enum delays {
  EVADE_TURN_DELAY=300,
  EVADE_STRAIGHT_DELAY=300,
  LIFT_DELAY = 900,
};
// distances for the sonar
enum distances {
  DETECT_DISTANCE=40,
  ENGAGE_DISTANCE=4,
};
// Sonar is on pins 12(PB4) and 13(PB5)
// PCINT4 PCINT5

// function declarations
void end_lift(void);
void turn_on_led(void);
void turn_off_led(void);
char read_switch(void);

// transistions/ states for out state machine
enum action {
  SEARCH_LEFT,
  SEARCH_RIGHT,
```

```
  EVADE,
  ENGAGE,
  FOUND,
  QTI_TRIGGERED,
  NONE,
};

// move straight while the robot is visible in both sonars
// go to engage if it is close
// if it is lost go back to search, with the direction it was last seen in
enum action pursue(void) {
  while(1) {
    float distleft = left_sonar();
    float distright = right_sonar();
    if (QTI_TRIPPED) {
      return QTI_TRIGGERED;
    } else if ((distleft<ENGAGE_DISTANCE) && (distright<ENGAGE_DISTANCE)){
      return ENGAGE;
    } else if ((distleft<DETECT_DISTANCE) && (distright<DETECT_DISTANCE)){
      move_motors(straight);
    } else if (distright < DETECT_DISTANCE) {
      return SEARCH_RIGHT;
    } else if (distleft< DETECT_DISTANCE) {
      return SEARCH_LEFT;
    }
  }
}

// spins while polling sonars
// pursues once other bot is 'seen'
enum action search(enum action d) {

  if (d==SEARCH_LEFT) {
    move_motors(sleft);
  } else {
    move_motors(sright);
  }

  while(1) {
    if (QTI_TRIPPED) {
      return QTI_TRIGGERED;
    }
    float distleft = left_sonar();
    float distright = right_sonar();
    if ((distleft<DETECT_DISTANCE) && (distright<DETECT_DISTANCE)){
      return FOUND;
    }
  }

}

// Takes actions to move off line, does not return until there are no
// lines detected.
enum action evade(void) {
  enum action action = EVADE;
  end_lift();
```

```
90    move_motors(stop);
      while(1) {
92      if (BRIGHT_QTI && BLEFT_QTI) {
          move_motors(straight);
94        action = SEARCH_LEFT;
        }
96      else if (FRIGHT_QTI) {
          move_motors(sleft);
98        action = SEARCH_LEFT;
        }
100     else if (BRIGHT_QTI) {
          move_motors(mright);
102       action = SEARCH_RIGHT;
        }
104     else if (FLEFT_QTI) {
          move_motors(sright);
106       action = SEARCH_RIGHT;
        }
108     else if(BLEFT_QTI) {
          move_motors(mleft);
110       action = SEARCH_LEFT;
        } else {
112       break;
        }
114   }

116   int time_left = EVADE_TURN_DELAY;
      while(time_left > 0) {
118     setdelay(10);
        waitfordelay();
120     time_left-=10;
        if (QTI_TRIPPED) {
122       return EVADE;
        }
124   }
      move_motors(straight);
126   time_left = EVADE_STRAIGHT_DELAY;
      while(time_left > 0) {
128     setdelay(10);
        waitfordelay();
130     time_left-=10;
        if (QTI_TRIPPED) {
132       return EVADE;
        }
134   }
      return action;
136
}
138

140 void start_lift(void) {
      PORTB |= (1<<PB1);
142   PORTB &= ~(1<<PB0);
}
144
    void end_lift(void) {
```

```
146   PORTB &= ~(1<<PB1);
      PORTB &= ~(1<<PB0);
148 }

150 void start_lower(void) {
      PORTB |= (1<<PB0);
152   PORTB &= ~(1<<PB1);
    }
154
    void end_lower(void) {
156   PORTB &= ~(1<<PB1);
      PORTB &= ~(1<<PB0);
158 }

160 void init_led(void) {
      DDRD |= (1<<PD1);
162   PORTD &= ~(1<<PD1);
    }
164
    void turn_on_led(void) {
166   PORTD |= (1<<PD1);
    }
168
    void turn_off_led(void) {
170   PORTD &= ~(1<<PD1);
    }
172
    void end(char q) {
174   if (CENTER_QTI) {
        turn_on_led();
176     end_lift();
        move_motors(stop);
178     while(1);
      }
180 }

182 // Lifts front flipper and pushes robot straight until QTI is detected,
    // when it passes into evade, and lowers front flipper.
184 enum action engage(void) {
      move_motors(straight);
186   enum action ret = NONE;
      int tl = 0;
188   start_lift();
      int time_left = LIFT_DELAY;
190   while(time_left > 0) {
        setdelay(10);
192     waitfordelay();
        time_left -= 10;
194     while (QTI_TRIPPED) {
          ret = EVADE;
196       while (ret == EVADE) {
            ret = evade();
198       }
        }
200     start_lift();
      }
```

```
202   end_lift();
      time_left = LIFT_DELAY;
204
      while(!(QTI_TRIPPED)) {
206     setdelay(10);
        waitfordelay();
208   }

210   move_motors(stop);
      ret = EVADE;
212   while (ret == EVADE) {
        ret = evade();
214   }

216   start_lower();
      while(read_switch() && time_left > 0) {
218     setdelay(10);
        waitfordelay();
220     time_left -= 10;
        while (QTI_TRIPPED) {
222       ret = EVADE;
          while (ret == EVADE) {
224         ret = evade();
          }
226     }
        start_lower();
228   }
      end_lower();
230   return ret;
}
232
void
234 init_switch(void) {
      DDRD &= ~(1<<PD7);
236   PORTD |= (1<<PD7);
}
238
char
240 read_switch(void) {
      char ret = PIND & (1<<PD7);
242   return ret;
}
244
int main(void) {
246   init_timer0();
      init_motors();
248   init_switch();
      init_qti(end);
250   init_led();
      DDRB |= (1<<PB0) | (1<<PB1);
252   init_sonar();
      sei();
254   move_motors(stop);

256   enum action dir = SEARCH_LEFT;
      while(1) {          // This loop directs actions into states for
```

```
258     switch (dir) {   // the duration of the match
          case FOUND:
260         dir = pursue();
            break;
262       case QTI_TRIGGERED:
            dir = evade();
264         break;
          case ENGAGE:
266         dir = engage();
            break;
268       case SEARCH_LEFT:
          case SEARCH_RIGHT:
270         dir = search(dir);
            break;
272       default:
            dir = SEARCH_LEFT;
274         break;
      };
276   }
}
```