🚗 ECE 302 Final Report

1. Information

- **Bench Number: 409**
- **Name(s)**: David Lee (dl1506@princeton.edu), Seungho (John) Lee (jl7339@princeton.edu)
- **Date**: 12/17/2025

2. Statement of Objectives

---

1. For our final project, we wanted to add to our existing car by making it track an object, capture the object and bring it to an objective point. This led to the creation of our SonnyBot (named after the soccer player Son Heung Min #7), which can search for a ball, track a ball once found, capture the ball, and kick it once it searches and finds the goal. It then does a little wiggle celebration after the goal is scored!

3. Initial Proposal

Our initial proposal was as follows:

"We aim to design a soccer robot car that operates autonomously within a defined boundary, similar to a soccer field. The field will be marked with black tape, measuring 10 meters by 5 meters. Our robot will use omnidirectional omniwheels to enable smooth and flexible movement in all directions. It will also feature a claw mechanism for ball retrieval and a spring-loaded launcher for shooting.

A mini soccer ball will be placed at a random location within the boundary. Using a trained Pixy camera, the robot will detect and navigate toward the ball. Once it is properly aligned for retrieval, the claw will lower and pull back, compressing the spring. In this loaded state, the robot will move toward the goal area. The Pixy camera will identify the two goalpost markers and help position the robot at an optimal shooting angle. Finally, when aligned, the claw will lift, releasing the compressed spring and launching the ball toward the goal."

So how close did we get to this proposal? While we did not end up using omniwheels for our car, we did build a H-bridge as explained in section 4 to control the forward and backward directions of the car. Furthermore as explained in section 6, we built a gate mechanism for ball retrieval instead of a claw and a rotating kicker for shooting instead of the compressed spring. In our initial proposal, we had the ball stationary but in a real life game of soccer, a ball moves around. As such, we were able to have the car track and follow the ball as it moved around and also identify the goal as well. Overall, the car was able to complete the main tasks that we wanted to achieve, but mechanically, it looked a little different than our initial planning!

4. Overview of Key Subsystems and Components

[Used the same car as speed control, see report for initial car implementation]

- We use Pixy2 on a pan tilt to search and track the objects that we have trained it to find. Pixy2 is smaller, faster and more capable than the original Pixy. Like its predecessor, Pixy2 can learn to detect objects that you teach it, just by pressing a button. It operates at 60 frames per second so we are able to control the car at a high frequency. The servo motors that control the pan tilt are plugged into the Pixy2 circuit on its back and are powered through the connection cable with the Arduino Uno. The Pixy2 is placed on a lasercutted extension of the car.
- The H-bridge is an electronic circuit used primarily for controlling DC motors to allow them to spin in both forward and reverse directions, and for precise speed control using Pulse Width Modulation (PWM). It works by using four switches (often transistors and in our case we use two NMOS and four PMOS transistors to more effectively control the current) to reverse the voltage polarity across the motor, effectively changing its rotation direction. The H-bridge is connected to the motor switch, which powers 7.2 V and two PWM signals coming from the PSOC (the signals from PSOC are 5V which we translate to 7.2 V using a logic level shifter).

- A logic level shifter is an electronic circuit that translates digital signals between different voltage levels. In this system, the microcontroller outputs 5 V PWM signals, but the H-bridge transistors require a higher logic voltage to fully turn on. The logic level shifter converts the 5 V control signals to a higher voltage compatible with the H-bridge, ensuring proper transistor switching while the motor itself is powered separately at 7.2 V.
- We use servo motors to build a contraption device to capture the ball using a gating mechanism, and to build a kicking mechanism to kick the ball. Servo motors are electromechanical devices that provide precise control over angular or linear position, speed, and acceleration, using a closed-loop feedback system with a motor and controller to constantly adjust movement to a desired setpoint. More details of the parts will be discussed in Section 6 Mechanical Breakdown.
- We use an Arduino Uno as one of our microcontrollers to control Pixy2 movement using PID control and to rotate all the servo motors. The Uno also sends signals to the PSOC, indicating which direction the car should move based on the Pixy2 movement and at what task stage the car is currently at (e.g., search ball stage and search goal stage). The power and ground pin from the Arduino has been extended in the circuit board on the top left of figure 1 for the servo motors connecting the kicker and the gate. The wires with the command pulses also connect the servo motors to their respective pins on the Arduino through the board.
- The car has an extended rectangular surface that has been laser cutted to place the pan tilt Pixy2 and fit the servo motors needed. All other parts (like the 409 top and the bumpers with the 7) are made of cardboard and construction paper as an aesthetic (but functional) design choice to our car as the bumpers prevent the ball from rolling underneath the car.
- From the previous car, we use the power distribution and the PSOC.

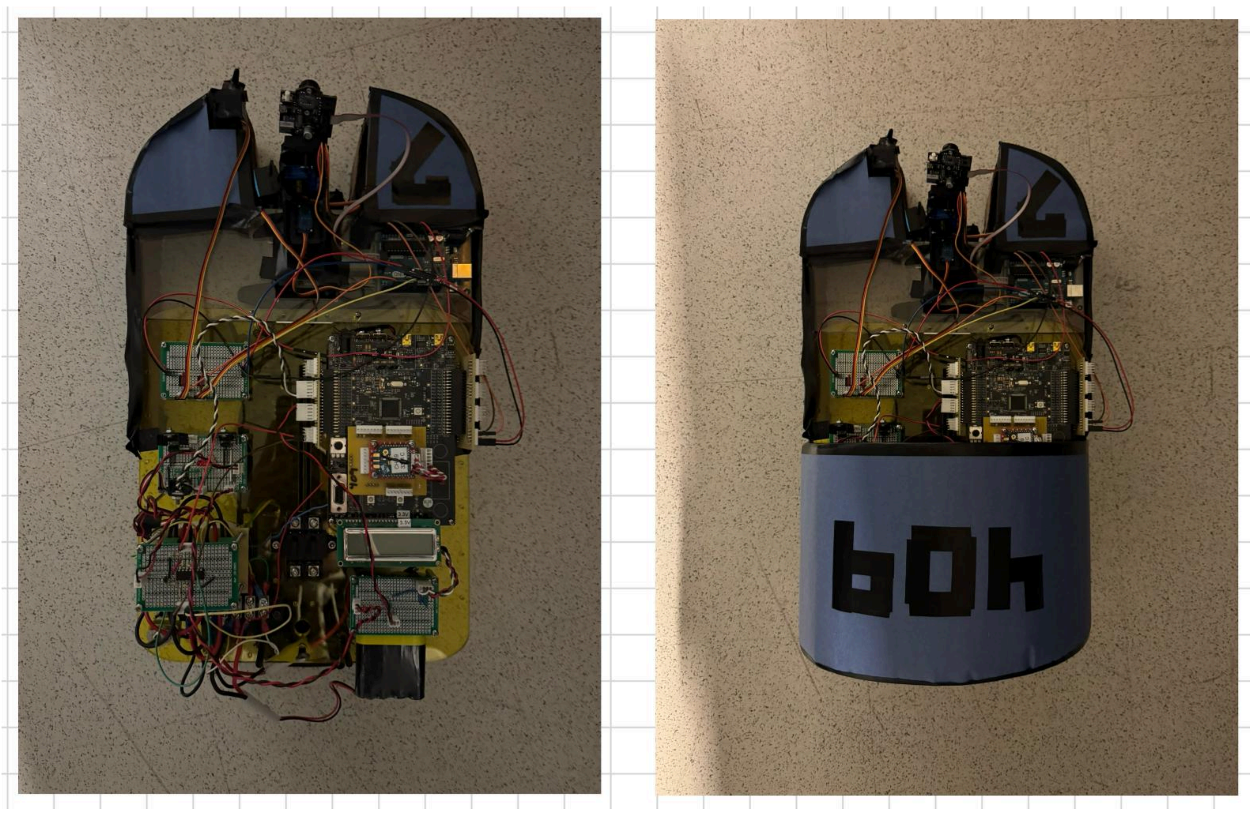


Fig. 1: Top down view of SonnyBot with the cover off (left) and cover on (right).

5. Schematic Diagram of Circuits

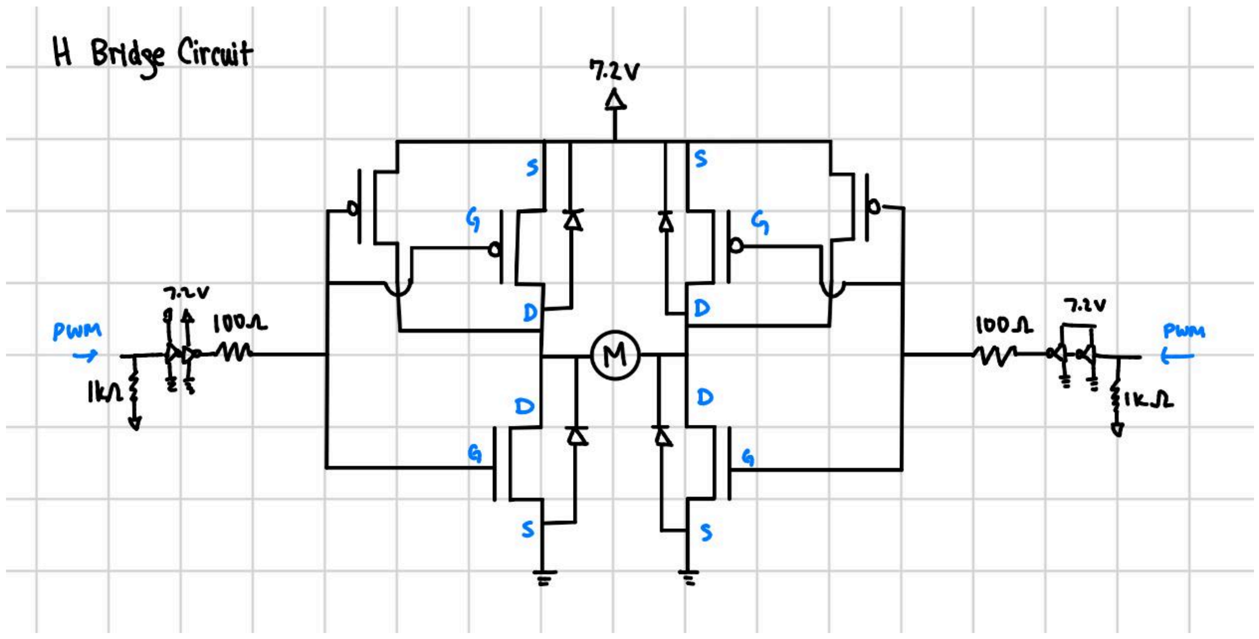


Fig. 2: Circuit diagram of the H-Bridge and the logic level shifter that shifts the 5V PWM signal from PSOC to 7.2 V. Four PMOS transistors (two in parallel on each side) are used to effectively control the current and prevent overheating, and two NMOS transistors are used to build the H-bridge.

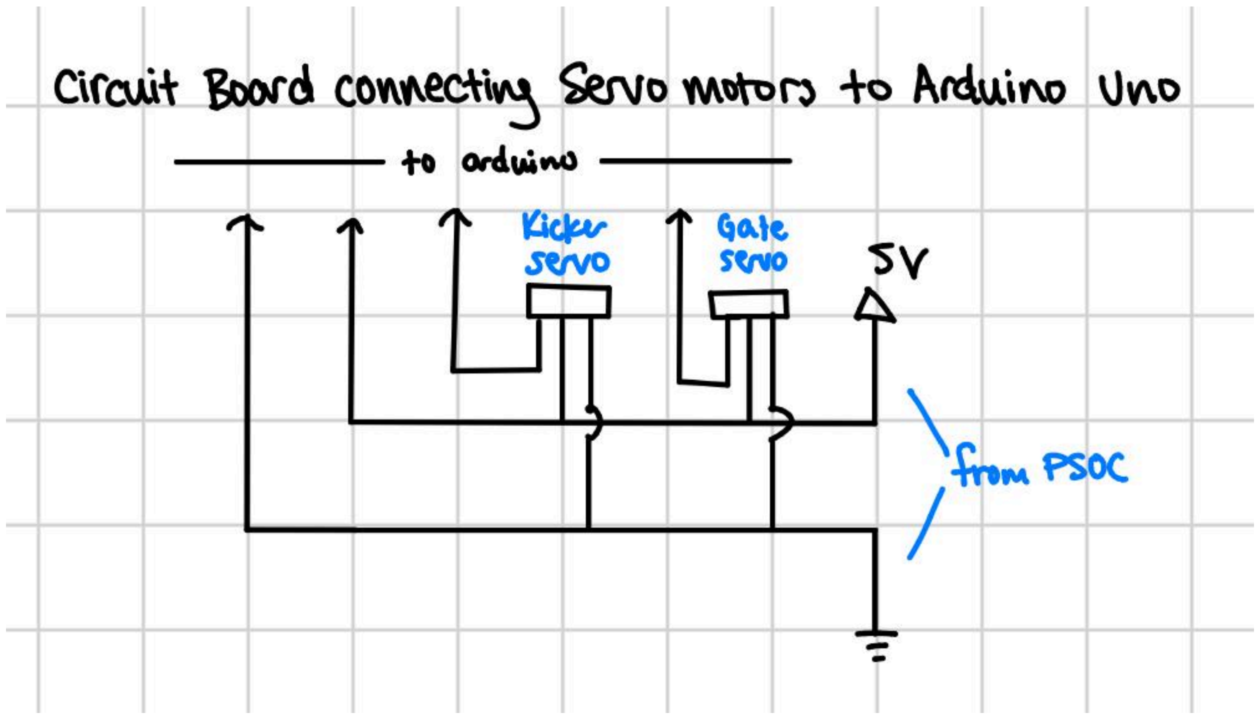


Fig. 3 Simple circuit diagram of board that powers Arduino Uno and the servo motors with 5V. Connects the servo motors to the Uno pins. 5V and the ground comes from the PSOC 5V supply rail and ground rail so that the arduino turns on when the PSOC turns on when the switch is flipped.

6. Mechanical Breakdown

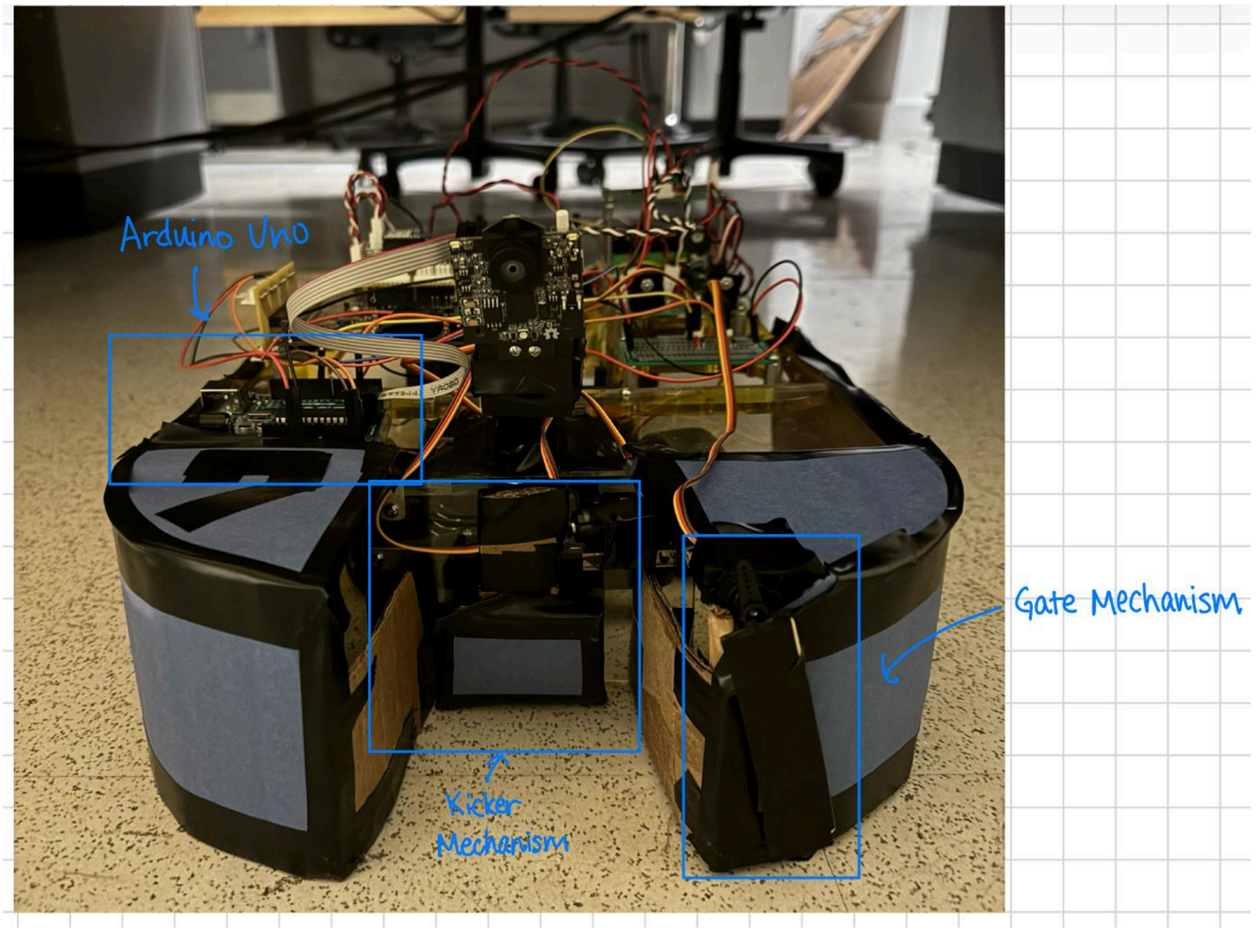


Fig. 4 Front side view of SonnyBot

Originally, we had planned to use a claw to capture our objective ball. However, because the claw did not arrive on time we had to improvise and build a new contraption using some cardboard that could capture the ball. Before the bumpers were added to the car, it was initially a fork of two pieces of cardboard that jutted out of the lasercutted extension, which the Pixy2 sat on. However, when the car moved around, we found it was not guaranteed the ball would always stay in between the two forks of cardboard. So, we added the gating mechanism to prevent the ball from rolling out once the car found the ball to mimic possessive "dribbling" rather than grabbing the ball because this is not handball, it's soccer. That way, once the car possessed the ball, it did not have to worry about losing it and could focus on its next objective, finding the goal.

We also added bumpers around the car so that the ball would not get trapped underneath the car and prevent the car from losing sight of the ball if it got too near and rolled underneath the car. This helped add a nice artistic touch to the car!

Once the Pixy2 tracked the location of the goal and the car was at a reasonable distance from the goal, we would lower the gate, and use the kicking mechanism to shoot the ball toward the goal. Other similar cars do not incorporate this kicker and we wanted to replicate a soccer player as realistically as we could. Unfortunately, the car cannot do a bicycle kick like Ronaldo but it will kick accurately.

Because the car does not rely on any sensors to spatially track whether it has captured the ball or it is near a goal, we use the information provided by the Pixy2 such as the size of the objects, and the angles of the pan tilt to provide signals to the car on how to operate. For example, to know whether the ball has been captured, the camera will track the trained ball and follow it until it is near the ball. Then when the angle of the pan is straight and the angle of the tilt is within a certain range (where Pixy2 is looking downward in the center between the bumpers) for a set time, we know the ball is in the right position for the gating mechanism to trap it.

Once this is done, we can move to the next task of searching for the goal. As the camera tracks the goal and moves near, it will compute the size of the goal if found. If the size of the goal meets the threshold, then we know that the car is near the goal and thus we can now kick the ball in the direction of that goal, and score!

We will explain more of our algorithm of how we are sending signals to the car through Arduino and PSOC in the code section below.

7. Data

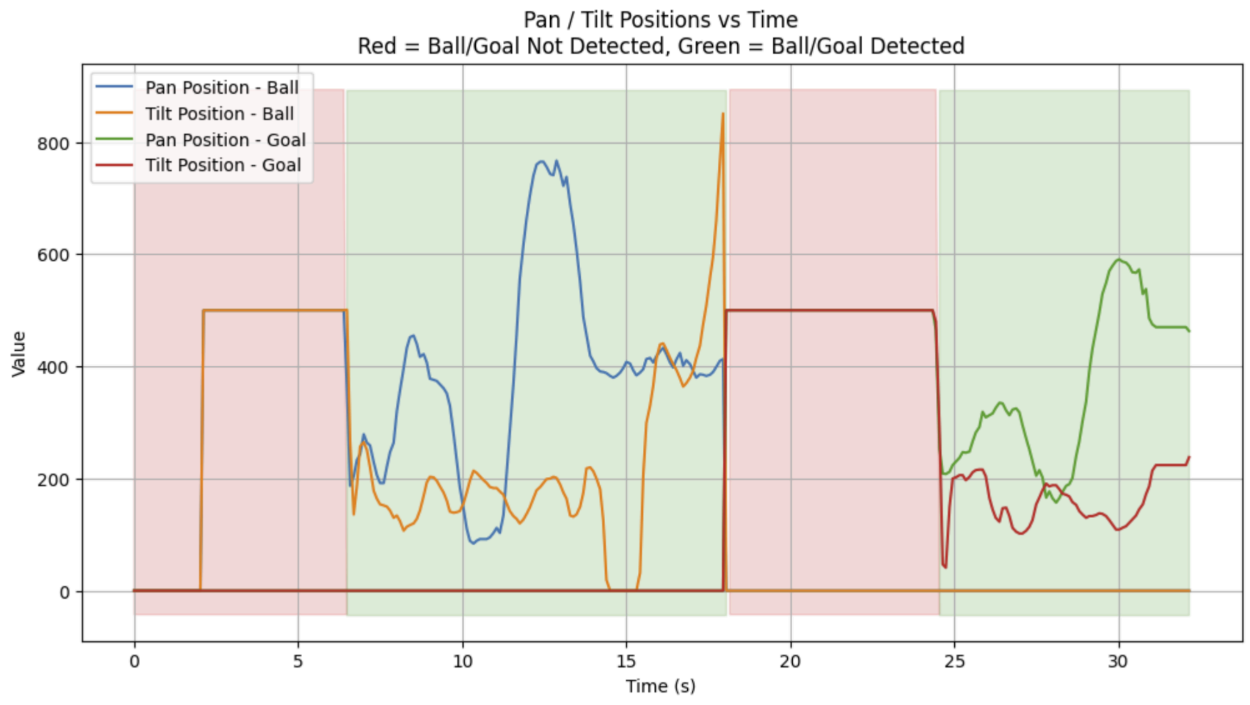


Fig. 4. All data transmitted from Arduino to PSoC via I2C over a certain period of time.

The figure above contains all data (either implicitly or explicitly) transmitted from the Arduino to the PSoC via I2C. The pan and tilt values shown in the plot correspond to raw position outputs from the Pixy camera, expressed in the Pixy's internal coordinate units rather than physical angles. These values serve as relative indicators of the target's position within the camera's field of view and typically span a range on the order of 0 to 1000, with a nominal midpoint near 500 depending on the specific mounting and calibration.

The first red shaded region indicates that the ball is not detected. During this phase, the pan mechanism oscillates left and right to search for the ball, and an arbitrary value of 500 is transmitted to the PSoC for both the pan and tilt positions. This value indicates that no valid target position is available, and therefore the pan and tilt positions are irrelevant to the PSoC while the ball is not detected. In our setup, the calibrated center values were approximately 400 for the pan axis and 200 for the tilt axis; however, the value of 500 was chosen solely as a placeholder and is not interpreted as a physical position.

The subsequent green region indicates that the ball has been detected. In this state, the pan position (blue line) and tilt position (orange line) track the ball, and these values are sent to the PSoC. Once the tilt position has exceeded 730 for over 0.3 seconds, then that represents the ball is captured and the Pixy camera transitions to searching for the goal.

The red region following capturing the ball indicates that the goal is not detected. As in the earlier search phase, the pan oscillates left and right to search for the goal, and an arbitrary value of 500 is again sent for both the pan and tilt positions for the same reason described above. The final green region indicates that the goal has been detected. During this phase, the pan position (green line) and tilt position (red line) track the goal and are transmitted to the PSoC.

Different line colors are used to distinguish between ball-tracking and goal-tracking tasks. This allows us to determine from the plot whether the car is still following the ball or has transitioned to following the goal in cases where the Pixy camera temporarily loses detection and then reacquires the target. Specifically, if the red shaded region is accompanied by blue and orange traces at a value of 500, the system is still searching for or following the ball. In contrast, if the red shaded region is accompanied by green and red traces, the system is searching for or moving toward the goal.

Once the car is sufficiently close to the goal, it shoots the ball and sends a signal to the PSoC indicating that the ball has been shot. This event is represented by the final data point in the plot, after which the graph terminates.

8. Challenges & Improvements

One of the main challenges of the car was figuring out how to capture a small moving ball without a claw. We first created and attached a small cardboard cutout onto the front of our car that nicely fit our ball. However, that was not enough as the ball often escaped after being captured. To fix this problem, we also added a gate mechanism in front of the cutout to prevent the ball from escaping once captured. This gate would close once the ball has been captured and open a couple deciseconds before the kicker mechanism is activated.

We were also faced with an issue of the ball getting stuck under our car. In order to prevent this, we cut and attached cardboard pieces as a “shield.” This not only prevented the ball from going under the car, but also improved the design of our car. As a result, we decided to add more cardboard pieces to our car to improve the design of our car.

Another challenge we faced was finding the right colors and lighting of the ball and the goal. We initially trained the Pixy camera to detect a red ball and blue goal. However, the Pixy camera also detected our skin colors as a red ball and it did not detect the blue goal very well. In addition, the performance of the Pixy camera would vary based on the lighting conditions. In order to minimize these problems, we made several improvements. First, we found out how to turn on flashlights from the Pixy camera which resulted in more versatility across various lightings. In addition, we decided to use a bright green ball and a bright yellow goal with black and white boards behind in order to increase contrast. This change allowed the Pixy2 to detect both the ball and the goal better.

We encountered some hardware challenges. The main issue was having to debug why our H bridge circuit burned. This was unexpected and surprising as the car had been working without issues for a while before the H-bridge suddenly failed and began smoking. We eventually found out that the transistors on one side of the H bridge stopped functioning correctly so we fixed that issue by replacing the NMOS and PMOS transistors on that side with new ones.

Lastly, on the software side, we had an issue with adjusting the steering constant. The steering constant served as how strongly the wheels steered based on the position of the pan. We noticed that we had to constantly change the duty cycle for the speed of the car and every time we did that, we also had to adjust the steering constant accordingly. If we increased the duty cycle for the speed of the car, we would have to decrease the steering constant. If we decreased the duty cycle, we would have to increase the steering constant. This was one of the main reasons why our car did not work as we expected during the final demonstration; we forgot to decrease the steering constant after increasing the duty cycle. This caused our car to oscillate too much which made it harder for our car to capture the ball.

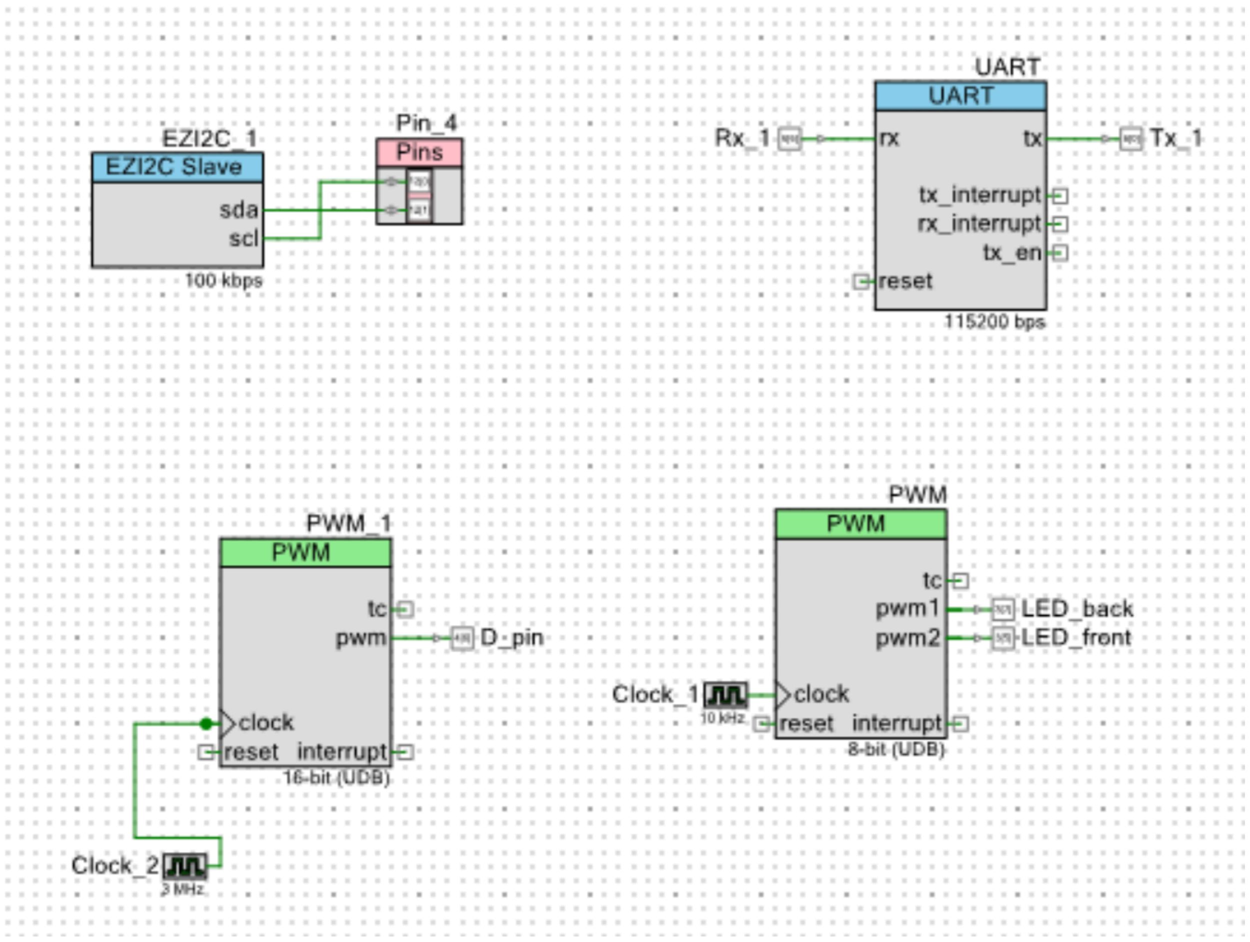
To improve our car in the future, we would like to mount the Pixy2 higher such that it has a greater scope of the environment and see the green ball and be able to track it at a greater range. We would also like to use a better camera that can track objects without having to worry about the brightness of the room or the color of the object. In a real soccer environment, there are many variables at play and vision is very important. Being able to track a ball that has not been trained in a testing environment but can still function in the testing environment will allow the car to participate in unseen environments with different lighting conditions. Furthermore, we would also like it to detect obstacles in the future and how to maneuver around them in pursuit of the ball or goal, which can probably be done using a sensor that detects distance of objects. This would mimic player to player interactions and add a layer of complexity to the car. Hopefully, we can also add edge detection (there were too many black lines in the lab to incorporate this) but given a rectangular field environment, the car will know to stay in bounds so the ball does not go out of play.

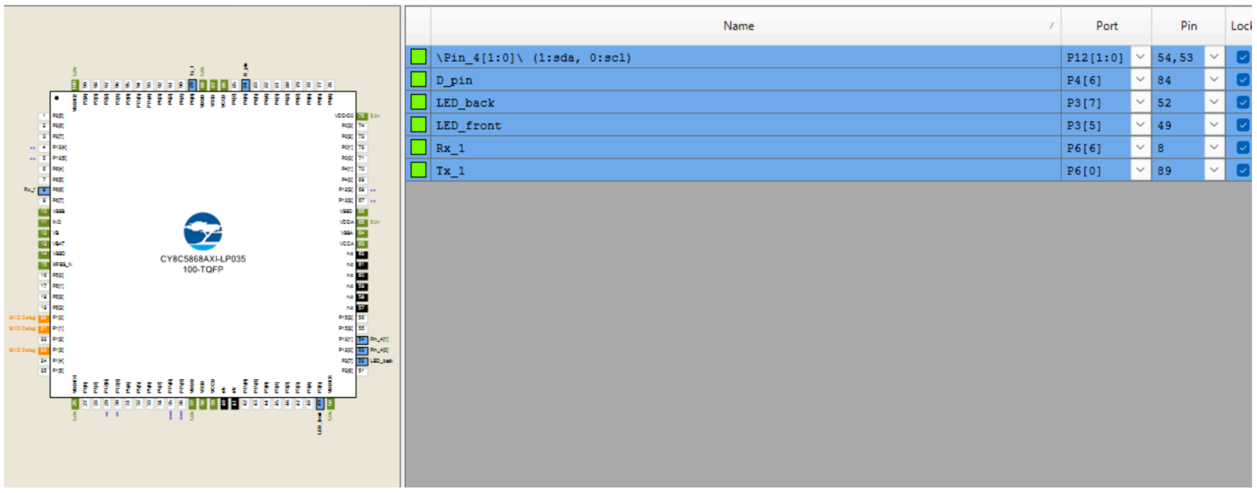
9. Test Run Video

---

https://drive.google.com/file/d/1lUulacU_20d38iS3vzKaKxBgQ2DpgnrC/view?usp=sharing
In the video, you will see the car do multiple tasks. Initially, the car does not see the ball so it will search for it by sweeping backward and rotating the pan tilt to find the ball. Once the ball is tracked it now enters the tracking stage where it attempts to catch up to the ball and capture it. We see that in the video when it loses sight of the ball in the frame of Pixy2 it will go back to the backward search sweep. After the ball is captured, the gate mechanism closes and traps the ball. The car now enters a second searching mode, looking for the goal through a forward sweep with the pan tilt rotating once more. In the video, we see that the car does one spin because it does not initially see the goal. But once it is locked on, the car will follow the goal until it is near enough, release the gate, and then shoot the ball with the kicker mechanism, and then stop. Task complete! Note that when this video was taken, we had not added the little celebration at the end.

10. Top Design





	Name	Port	Pin	Lock
🟩	\Pin_4[1:0]\ (1:sda, 0:scl)	P12[1:0]	54,53	✓
🟩	D_pin	P4[6]	84	✓
🟩	LED_back	P3[7]	52	✓
🟩	LED_front	P3[5]	49	✓
🟩	Rx_1	P6[6]	8	✓
🟩	Tx_1	P6[0]	89	✓

Summary:

- The EZI2C Slave component receives data from Arduino
    - SDA is the wire that carries the bits of the data
    - SCL is generated by Arduino which synchronizes the Arduino and the PSoC and makes sure bits are read at the correct time

    - This is set to 100 kbps (100,000 bits per second) which is the data rate of the I2C bus.
    - The Pins component attached connects signals inside the PSoC to the physical pins on the PSoC hardware
- The UART component is a serial communication protocol used for debugging and logging
    - It sends bits one a time over a single wire
    - Unlike I2C, it has no shared clock and so the timing is inferred from baud rate
        - We have the baud rate set to 115200 which means it transmits/samples 115200 bits per second
        - Note that if we want to change the baud rate for whatever reason, we also have to change it on the receiving side (we have the code in Python)
        - rx and tx are connected to Rx_1 and Tx_1 pins, respectively
- The PWM components drive actuators
    - PWM controls the speed for both going forward and backward
        - We set the PWM mode to 2 outputs as we are using the H bridge for our car
            - pwm1 is connected to the pin representing LED_back and it controls how fast the car moves back
            - pwm2 is connected to the pin representing LED_front and it controls how fast the car moves forward
            - We were very careful not to set both of the pwms at once. At least one output had to be set to 0 in order to prevent errors in our car
    - PWM_1 controls the direction of the wheels
        - We set the clock frequency to 3 MHz. This means the clock period is ~0.3333 µs per count
        - Based on the lecture, when the steering angle is -90 degrees (full left), the pulse width is 1 ms. When it's 0 degrees (center), the pulse width is 1.52 ms. When it's 90 degrees (full right), the pulse width is 2 ms. Using this, we can calculate the compare value:
            - Compare value = pulse width / clock period (to simplify, this basically means the number of ticks equals the time you want divided by the time per tick)
            - Using the equation, for full left, compare value will be 3000. For middle, it'll be 4560. For full right, it will be 6000
            - We set the period to 6000 because that is the maximum number the compare value will get

        - We noticed that for our car, 6000 as the compare value did not work very well so we set it a little lower to 5990 as the value for full right.
    - The pwm is connected the pin representing D_pin

11. Code

Arduino:

```
//
// begin license header
//
// This file is part of Pixy CMUcam5 or "Pixy" for short
//
// All Pixy source code is provided under the terms of the
// GNU General Public License v2
(http://www.gnu.org/licenses/gpl-2.0.html).
// Those wishing to use Pixy source code, software and/or
// technologies under different licensing terms should contact us at
// cmucam@cs.cmu.edu. Such licensing terms are available for
// all portions of the Pixy codebase presented here.
//
// end license header
//

#include <Pixy2.h>
#include <PIDLoop.h>
#include <Wire.h>
#include <Servo.h>

Pixy2 pixy;
// Adjust Proportional, Integral, and Derivative controls to keep the
pan and tilt stable
PIDLoop panLoop(300, 100, 500, true);
PIDLoop tiltLoop(500, 0, 500, true);
```

```
int panPos = 400; // Center value for our pan
int panDir = 1; // Direction the pan rotates in
int panMin = 200; // Minimum position that the pan moves when searching
int panMax = 600; // Maximum position that the pan moves when searching

// Variables for oscillating the pan when object not detected
unsigned long lastMove = 0;
int stepDelay = 5;
int stepSize = 5;

Servo servo1; // Shooting mechanism
Servo servo2; // Gate that captures the ball and prevents it from
escaping
bool s = false; // Shot or not

unsigned long lastTimeTiltHigh = 0; // Time captured when tilt is facing
down (ball is captured)
int k = 1; // k=1 when part 1 and k=2 when part 2


const byte PSOC_ADDR = 0x08; // Match with slave address

// Helper to send data to PSoC over I2C
void sendToPsoc(bool hasObject, int16_t panCmd, int16_t tiltCmd, int16_t
panCmd2, int16_t tiltCmd2, bool shoot) {
  Wire.beginTransmission(PSOC_ADDR);

  Wire.write((uint8_t)0); // Dummy because in our case, the PSoC started
reading from the 2nd byte

  // Byte 0: ball detected or not
  Wire.write(hasObject ? 1 : 0);

  // Byte 1-2: pan position for part 1
  Wire.write((panCmd >> 8) & 0xFF);
  Wire.write(panCmd & 0xFF);

  // Byte 3-4: tilt position for part 1
  Wire.write((tiltCmd >> 8) & 0xFF);
  Wire.write(tiltCmd & 0xFF);

  // byte 5-6: pan position for part 2
  Wire.write((panCmd2 >> 8) & 0xFF);
  Wire.write(panCmd2 & 0xFF);

  // byte 7-8: tilt position for part 2
  Wire.write((tiltCmd2 >> 8) & 0xFF);
  Wire.write(tiltCmd2 & 0xFF);

  // byte 9: ball shot or not
  Wire.write(shoot ? 1 : 0);

  Wire.endTransmission();
}

void setup()
{
  Serial.begin(115200);
  Wire.begin();
  pixy.init();
  pixy.changeProg("color_connected_components");
  pixy.setLamp(1,1);

  servo1.attach(9);
  servo1.write(0);

  servo2.attach(3);
  servo2.write(120);
}
```

```
void loop()
{
  // Part 1: capturing the ball
  if (k == 1) {
    int32_t panOffset, tiltOffset;

    // Get the ball detected from pixy
    pixy.ccc.getBlocks(true, CCC_SIG1);

    // Pixy cam detects the ball
    if (pixy.ccc.numBlocks)
    {

      // Calculate pan and tilt "errors" with respect to first object
(blocks[0]),
      // which is the biggest object (they are sorted by size).
      panOffset = (int32_t)pixy.frameWidth/2 -
(int32_t)pixy.ccc.blocks[0].m_x;
      tiltOffset = (int32_t)pixy.ccc.blocks[0].m_y -
(int32_t)pixy.frameHeight/2;

      // Update loops
      panLoop.update(panOffset);
      tiltLoop.update(tiltOffset);

      // Set pan and tilt servos
      pixy.setServos(panLoop.m_command, tiltLoop.m_command);

      // Send information to psoc
      sendToPsoc(true, (int16_t)panLoop.m_command,
(int16_t)tiltLoop.m_command, 0,0,s);

      // If the tilt is facing down enough, close the gate to capture
the ball and move onto the second part
```

```
      if (tiltLoop.m_command > 730) {
        // If first time entering this condition, capture the time
        if (lastTimeTiltHigh == 0) {
          lastTimeTiltHigh = millis();
        }

        // If it has been above 730 for over 0.2 second, close the gate
        if (millis() - lastTimeTiltHigh >= 200) {
          k=2;
          lastTimeTiltHigh = 0;
          servo2.write(60);
        }
      }
      else {
        // Reset timer when tilt is not facing down enough (the ball is
not in our possession)
        lastTimeTiltHigh = 0;
      }


    }
    // Ball is not detected
    else
    {
      panLoop.reset();
      tiltLoop.reset();
      // Oscillate the pan (searching for the ball)
      if (millis() - lastMove > stepDelay)
      {
          lastMove = millis();
          panPos += panDir * stepSize;

          // Reverse direction at edges
          if (panPos >= panMax || panPos <= panMin)
              panDir *= -1;
      }
      // Tilt stays centered
      int tiltPos = 200;

      pixy.setServos(panPos, tiltPos);

      // Send information to psoc
      sendToPsoc(false, (int16_t)panLoop.m_command,
(int16_t)tiltLoop.m_command, 0, 0,s);
    }
  }




  // Part 2: scoring the goal
  if (k == 2) {
    int32_t panOffset, tiltOffset;

    // Get the goal detected from pixy
    pixy.ccc.getBlocks(true, CCC_SIG2);

    // The goal is detected
    if (pixy.ccc.numBlocks)
    {

      // calculate pan and tilt "errors" with respect to first object
(blocks[0]),
      // which is the biggest object (they are sorted by size).
      panOffset = (int32_t)pixy.frameWidth/2 -
(int32_t)pixy.ccc.blocks[0].m_x;
      tiltOffset = (int32_t)pixy.ccc.blocks[0].m_y -
(int32_t)pixy.frameHeight/2;

      // update loops
      panLoop.update(panOffset);
      tiltLoop.update(tiltOffset);

      // set pan and tilt servos
      pixy.setServos(panLoop.m_command, tiltLoop.m_command);

      // If the car is close enough to the ball, open the gate, and then
once the gate fully opens, shoot the ball
      if (pixy.ccc.blocks[0].m_height > 60 && pixy.ccc.blocks[0].m_width
> 180) {
        s = true;
        servo2.write(120);
        delay(500);
        servo1.write(120);
        delay(500);
      }

      // Send information to psoc
      sendToPsoc(true, 0, 0, (int16_t)panLoop.m_command,
(int16_t)tiltLoop.m_command, s);
    }
    // Goal is not detected
    else
    {
      panLoop.reset();
      tiltLoop.reset();
      // Oscillate the pan (searching for the goal)
      if (millis() - lastMove > stepDelay)
      {
          lastMove = millis();
          panPos += panDir * stepSize;

          if (panPos >= panMax || panPos <= panMin)
              panDir *= -1;
      }
      // Tilt stays centered
      int tiltPos = 200;

      pixy.setServos(panPos, tiltPos);

      // Send information to psoc
      sendToPsoc(false, 0, 0, (int16_t)panLoop.m_command,
(int16_t)tiltLoop.m_command,s);

      servo1.write(0);
    }


  }
}
```

PSoC:

```
/* ========================================
 *
 * Copyright YOUR COMPANY, THE YEAR
 * All Rights Reserved
 * UNPUBLISHED, LICENSED SOFTWARE.
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION
 * WHICH IS THE PROPERTY OF your company.
 *
 * ========================================
*/
#include "project.h"
#include <stdio.h>
#include <math.h>


// Pixy pan range: usually 0–1000; we found that our center is 400
```

```
#define PAN_CENTER 400

// Steering
#define SERVO_CENTER 4560 // Straight
#define SERVO_MIN 3000 // Left
#define SERVO_MAX 5990 // Right; we noticed that 6000 did not work as expected

// This affects how strongly pan error affects steering
#define STEER_GAIN 10 // adjust this whenever we change speed (higher speed --> lower steer gain)

// Buffer that stores information received from Arduino through I2C
/* 0: object seen or not, 1-2: pan when focused on ball, 3-4: tilt when focused on ball
5-6: pan when focused on goal, 7-8: tilt when focused on goal, 9: ball shot or not

From now on: part 1 is when the car is focused on the ball and part 2 is when the car is focused on the
goal*/
uint8 i2cBuffer[10];

volatile uint8 ballVisible = 0;
volatile int16 panCmd = 0;
volatile int16 tiltCmd = 0;
volatile int16 panCmd2 = 0;
volatile int16 tiltCmd2 = 0;
volatile uint8 shot = 0;



int main(void)
{
    CyGlobalIntEnable; /* Enable global interrupts. */

    /* Initializations */
    UART_Start();
    // Start I2C slave
    EZI2C_1_Start();
    EZI2C_1_SetBuffer1(sizeof(i2cBuffer), sizeof(i2cBuffer), i2cBuffer);

    // For speed
    PWM_Start();
    Timer_Start();

    // For navigation
    PWM_1_Start();

    for(;;)
    {
        // Description for each variable here described above in i2c buffer
        ballVisible = i2cBuffer[0];
        CyDelay(100); // 0.1 sec delay
        panCmd  = (int16)(( (int16)i2cBuffer[1] << 8 ) | i2cBuffer[2]);
        tiltCmd = (int16)(( (int16)i2cBuffer[3] << 8 ) | i2cBuffer[4]);
        panCmd2  = (int16)(( (int16)i2cBuffer[5] << 8 ) | i2cBuffer[6]);
        tiltCmd2 = (int16)(( (int16)i2cBuffer[7] << 8 ) | i2cBuffer[8]);
        shot = i2cBuffer[9];

        // UART
        char buf[32];
        sprintf(buf, "%d %d %d %d %d %d\r\n", (int) ballVisible, (int) panCmd, (int) tiltCmd, (int)
panCmd2, (int) tiltCmd2, (int) shot);
        UART_PutString(buf);

        // If ball is shot, do a celebration
        if (shot)
        {
            PWM_WriteCompare2(0); // For moving forward
            PWM_WriteCompare1(0); // For moving backward
            CyDelay(1000);
            PWM_WriteCompare1(80);
            PWM_1_WriteCompare(3000); // Wheels facing left
            CyDelay(1000);
            PWM_1_WriteCompare(5990); // Wheels facing right
            CyDelay(1000);
            PWM_1_WriteCompare(3000);
            CyDelay(1000);
            PWM_1_WriteCompare(5990);
            CyDelay(1000);
        }

        // Part 1: if the ball is detected, go towards the ball
        else if(ballVisible && !panCmd2 && !tiltCmd2)
        {
            PWM_WriteCompare2(60);
            PWM_WriteCompare1(0);

            // How far the pan is from center
            double panError = (double)panCmd - (double)PAN_CENTER;

            // Proportional control
            double compare = SERVO_CENTER - STEER_GAIN * panError;

            // Edge cases
            if (compare < SERVO_MIN) compare = SERVO_MIN;
            if (compare > SERVO_MAX) compare = SERVO_MAX;



            PWM_1_WriteCompare((uint16)compare);
        }

        // Part 1: if the ball is not detected, search for the ball (rotate moving back with the wheel facing
    left)
        else if(!ballVisible && !panCmd2 && !tiltCmd2)
        {
            PWM_1_WriteCompare(3000);
            PWM_WriteCompare1(60);
            PWM_WriteCompare2(0);
        }

        // Part 2: if the goal is visible, go towards the goal
        else if(ballVisible && !panCmd && !tiltCmd)
        {
            // Same code as part 1 when the ball is detected


            PWM_WriteCompare2(60);
            PWM_WriteCompare1(0);

            double panError = (double)panCmd2 - (double)PAN_CENTER;

            double compare = SERVO_CENTER - STEER_GAIN * panError;

            if (compare < SERVO_MIN) compare = SERVO_MIN;
            if (compare > SERVO_MAX) compare = SERVO_MAX;

            PWM_1_WriteCompare((uint16)compare);
        }

        // Part 2: if the goal is not visible, search for the goal (rotate moving forward with the wheel facing
    left)
        else
        {
            PWM_1_WriteCompare(3000);
            PWM_WriteCompare2(50); // Speed is slowed down as the pixy cam did not detect the goal as
    well
```

```
            PWM_WriteCompare1(0);
        }
    }
}


/* [] END OF FILE */
```