Final Project Report

1. Team Information (5 pts)

Include your group number, team members' names and email addresses.

Group 401

Daniel Jung, Yiying Zhang

danieljung@princeton.edu, yiying.zhang@princeton.edu

2. Project Objective (10 pts)

Our final objective was to create an automated laser turret that uses a pan/tilt mount and the existing car to follow, aim, and shoot at a specified target. Building upon some of the car's existing systems, we used a Pixy2 camera (or a webcam and OpenCV in earlier iterations) for object recognition, an Arduino Uno for input/output processing, and existing power/steering solutions rewired to the Arduino.

3. System Architecture (20 pts)

Provide a detailed overview of your project's key components and subsystems.

- **Arduino Uno:** The brain of our system, an Arduino microcontroller interfaces with the Pixy2—our object recognition sensor—to processes object location, rough size, and movement. It then outputs PWM signals that control turret movement, car steering, and speed. The Arduino draws power directly from the terminal block through the same wiring of the PSoC (now disconnected).
- **Moving Turret:** Two servos control pan & tilt, respectively. To ensure effective steering, pan movements are limited to a little less than 180 degrees (side to side) while the tilt angle is limited to 90 (forward and up). The turret is mounted on a pill bottle that also acts as an manual rotation adjustment base.

- **Laser:** A simple 5 mW laser diode was attached to the turret to activate when centered on a target. This laser was attached to the GND on the component board and received power from the Arduino.
- **Pixy2 Camera:** The Pixy camera allows for simple, yet effective hue-based recognition of different objects. While our initial design started out with a webcam running facial recognition through OpenCV, a Pixy2 worked much faster with much less processing power required.
- **Component Board:** An additional component board was needed to power the different servos and lasers. Powered with 5V from the voltage board, the board had 3 terminals to provide power to the servos and laser. A $0.1\ \mu F$ capacitor was placed before every terminal to control noise.
- **Voltage Board (from previous projects):** The voltage regulator board in our robotic car is equipped with two voltage regulators to manage power from a single 9.6V battery (BAT1). One regulator, a 7805 in the circuit diagram, steps down the 9.6V to 5V to power the component board. The second regulator converts the same 9.6V source to 6V to power the steering servo located on the same board.
- **Terminal Block:** The system draws all power from onboard batteries through the terminal block. The power wiring from the power distribution system in our robotic car utilizes two distinct battery sources: BAT1 with 7.2V and BAT2 with 9.6V. Each battery is connected through a dedicated line, providing the required voltages to different subsystems of the project. BAT1 powers the motor board directly via a 7.2V SPST switch. BAT2 is routed through a DPDT switch and distributes power to both the voltage and the Arduino (separated by said switch). The choice was made to use a DPDT switch vs a SPST switch in order to prevent ground loops between the Arduino, voltage regulator board, and the terminal block; if we used a SPST, the induced current loop may have caused latent current draw. The terminal block serves as a central node for all wiring connections, streamlining the organization and modification of power lines for enhanced maintenance and troubleshooting efficiency.
- **Steering Servo (Control on Voltage Board):** The original steering servo from the navigation report controls the front wheel steering. It now receives commands from the Arduino instead of PSOC. The servo receives signal

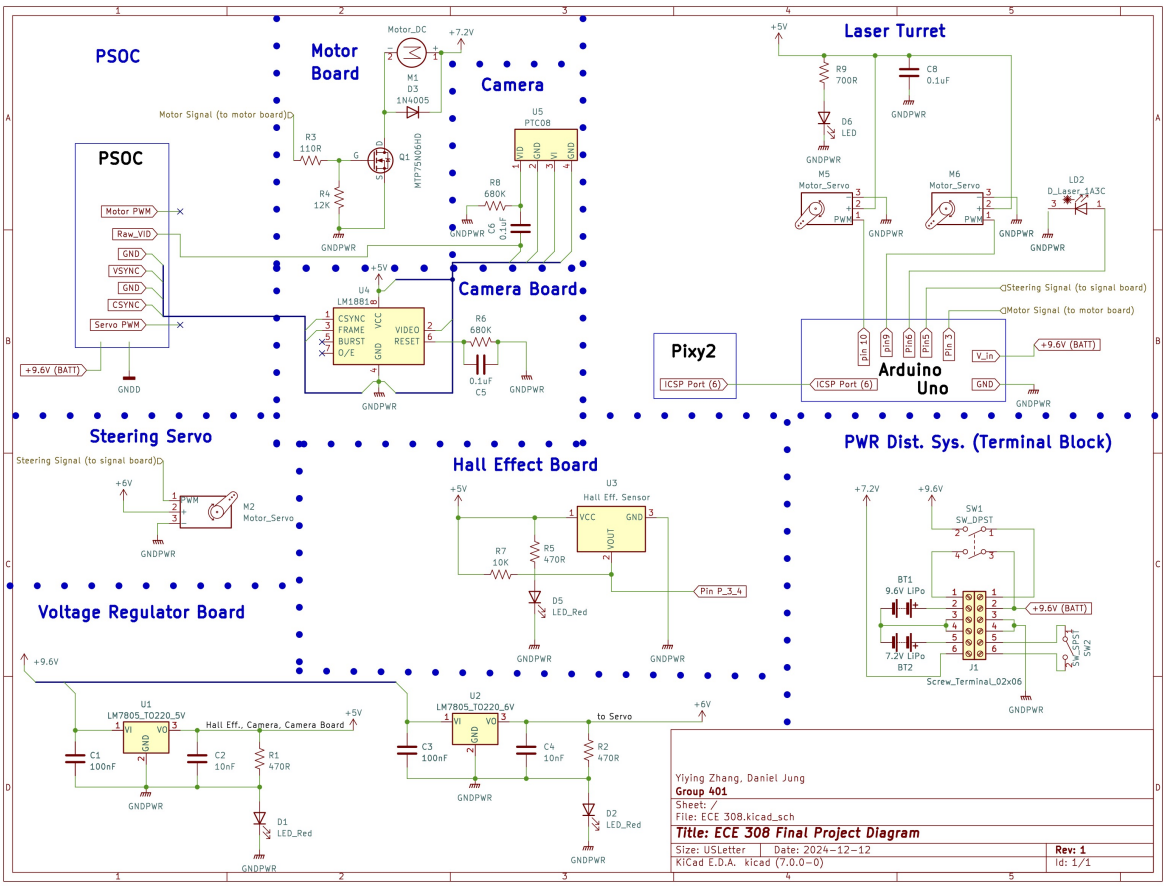
through an additional terminal on the power/voltage board that feeds it 6V, GND, and a PWM signal from the Arduino

- **Motor Board/Motor:** The original motor and control board from the navigation report controls the car wheels. They now receive commands from the Arduino instead of PSOC.

---

4. Circuit Documentation (20 pts)

Include clear, professional schematic diagrams of all circuits used.



> (Also attached to .zip file as “302 final proj diagram.pdf”)

5. Implementation & Results (20 pts)

Explain your design's functionality and include supporting data. A video demonstration is recommended.

> (See “demo1.mov” & “demo2.mov” video files attached in the .zip file)

Demo1.mov

The first demo video demonstrates the turret movement/targeting mechanism. As shown, the turret moves based on the location of the blue block. Once locked on, (object has been centered for a certain number of frames), the laser fires at the target. If the object is not to be found (out of frame) the turret turns from side to side in an effort to find the target.

Demo2.mov

The second demo video demonstrates turret movement with steering. In lockstep with the turret, the car is programmed to move towards the target (i.e. keep it in range) once the block moves far enough. It stops once the width of the object exceeds a certain percentage of the frame width.

Serial Monitor Log

The following is a serial monitor log we used while debugging. (Not shown in code due to high overhead/lag.)

```
Looking for target (horizontal panning)....
_________
Scanning: Looking to the left.
Scanning: Looking to the left.
Scanning: Looking to the right.
Scanning: Looking to the right.
Scanning: Looking to the right.
__________
Target located! X: 27 Y: 156
High speed adjustment: Target far from center.
Panning left to align with target.
```

```
Tilting down to track target.
Target not centered. Holding fire.
Adjusting position: Target within range.

Target not centered. Holding fire.
Moving forward: Target far away.

Target located! X: 179 Y: 74
High speed adjustment: Target far from center.
Tilting down to track target.
Target not centered. Holding fire.
Adjusting position: Target within range.

Target located! X: 84 Y: 139
High speed adjustment: Target far from center.
Panning left to align with target.
Tilting down to track target
.
Target not centered. Holding fire.
Moving forward: Target far away.

Target located! X: 176 Y: 77
High speed adjustment: Target far from center.
Tilting down to track target.

Target not centered. Holding fire.
Holding position: Target close enough.

Target located! X: 140 Y: 144
High speed adjustment: Target far from center.
Tilting down to track target.

Target not centered. Holding fire.
Holding position: Target close enough.
_____________________________
Target located! X: 241 Y: 165
```

```
Target locked! Laser fired.
Target locked! Laser fired.
Target locked! Laser fired.
Target locked! Laser fired.
Target locked! Laser fired.
Target locked! Laser fired.
Target locked! Laser fired.
```

Code

Arduino code is shown below, with comments describing different functions.

```
#include <Servo.h>
#include <Pixy2.h>

Pixy2 pixy;

// initialize servos
Servo panServo;
Servo tiltServo;
Servo steeringServo;

// define pins
const int panPin = 9;
const int tiltPin = 10;
const int motorPin = 3;
const int steeringPin = 5;
const int laserPin = 6;

// define default servo positions (pan)
const int panCenter = 100;
const int panRight = 10;
const int panLeft = 200;

// define default servo positions (tilt)
const int tiltCenter = 180;
```

```
const int tiltUp = 90;

// define default servo positions (steering)
const int steerCenter = 90;
const int steerLeft = 5;
const int steerRight = 175;

// define center of the frame
const int centerWidth = pixy.frameWidth / 2;
const int centerHeight = pixy.frameHeight / 2;

// get frame dimensions
int frameWidth = pixy.frameWidth;
int frameHeight = pixy.frameHeight;

// set initial movement speed, locking mechanism
int moveSpeed = 1;
int centerCounter = 0;
const int centerThreshold = 50; // # of frames to lock target

// set initial seeking direction
bool turnRight = true;

void setup() {
  // initialize pixy, serial communication
  Serial.begin(9600);
  pixy.init();
  pixy.setLamp(1, 1);

  // initialize servos
  panServo.attach(panPin);
  tiltServo.attach(tiltPin);
  steeringServo.attach(steeringPin);
  pinMode(motorPin, OUTPUT);
  pinMode(laserPin, OUTPUT);
```

```
  // set initial servo positions
  panServo.write(panCenter);
  tiltServo.write(tiltCenter);
  steeringServo.write(steerCenter);
}

void loop() {
  pixy.ccc.getBlocks();

  // if object found
  if (pixy.ccc.numBlocks) {
    // get object position + width
    int x = pixy.ccc.blocks[0].m_x;
    int y = pixy.ccc.blocks[0].m_y;
    int blockWidth = pixy.ccc.blocks[0].m_width;

    // calculate difference from center
    int xDiff = abs(x - centerWidth);
    int yDiff = abs(y - centerHeight);

    // adjust movement speed based on distance from center
    if (xDiff > 130 || yDiff > 100) {
      // object is far away
      moveSpeed = 3;
    } else if (xDiff > 100 || yDiff > 80) {
      // object is close
      moveSpeed = 1;
    }

    // adjust servos based on object position
    // frame is divided into thirds, with the center third being
    // X axis: left, center, right
    if (x < frameWidth / 3) {
      // object is left
      panServo.write(panServo.read() + moveSpeed);
      steeringServo.write(steeringServo.read() - moveSpeed);
```

```
      Serial.println("Moving left");
      centerCounter = 0;
    } else if (x > 2 * frameWidth / 3) {
      // object is right
      panServo.write(panServo.read() - moveSpeed);
      steeringServo.write(steeringServo.read() + moveSpeed);
      Serial.println("Moving right");
      centerCounter = 0;
    } else {
      // center
      centerCounter++;
    }

    // Y axis: up, center, down
    if (y < frameHeight / 3) {
      // object is up
      tiltServo.write(tiltServo.read() - moveSpeed);
      Serial.println("Moving up");
      centerCounter = 0;
    } else if (y > 2 * frameHeight / 3) {
      // object is down
      tiltServo.write(tiltServo.read() + moveSpeed);
      Serial.println("Moving down");
      centerCounter = 0;
    } else {
      // center
      centerCounter++;
    }

    // laser control is dependent on centerCounter
    // if object is centered for a certain amount of time, laser
    // otherwise, laser is turned off
    if (centerCounter >= centerThreshold) {
      // centered for a while (target locked)
      digitalWrite(laserPin, HIGH);
    } else {
```

```
      // not centered (off)
      digitalWrite(laserPin, LOW);
    }

    // motor control is dependent on blockWidth
    // if object is far away, motor is set to max speed
    // if object is close, motor is set to medium speed
    // if object is very close, motor is turned off
    if (blockWidth < (0.1 * frameWidth)) {
      // object is far away
      analogWrite(motorPin, 70);
    } else if (blockWidth < (0.2 * frameWidth)) {
      // object is close
      analogWrite(motorPin, 50);
    } else {
      // object is very close
      analogWrite(motorPin, 0);
    }
  } else {
    // IF NOT FOUND

    // Turn laser off
    digitalWrite(laserPin, LOW);
    centerCounter = 0;

    // Turn off motor
    analogWrite(motorPin, 0);

    // Look around
    tiltServo.write(150); // look a bit upwards
    if (turnRight) {
      // turn to the right
      panServo.write(panServo.read() + moveSpeed);
      if (panServo.read() >= 150) {
        turnRight = false;
      }
```

```
      Serial.println("Look right");
    } else {
      // turn to the left
      panServo.write(panServo.read() - moveSpeed);
      if (panServo.read() <= 30) {
        turnRight = true;
      }
      Serial.println("Look left");
    }
  }

  delay(20);
}
```

6. Technical Challenges (10 pts)

Discuss obstacles encountered and your design decisions.

Distance Sensor

Initially, we mounted a Time-of-Flight (ToF) distance sensor module beneath the Pixy2 camera on the turret assembly. This sensor could measure distances between 1 cm and 6 m, providing a precise line-of-sight measurement for determining the target's range. However, integrating it posed a memory challenge: the required sensor library exceeded the Arduino Uno's available memory.

To address this, we considered alternative microcontrollers. While the Arduino Mega offered sufficient static memory, it still lacked enough dynamic memory to properly run the required library. We then explored ARM M0-based microcontrollers, but they did not feature the Arduino-specific ICSP port, necessitating a cable modification that we aimed to avoid.

As another attempt, we switched to an HC-SR04 ultrasonic distance sensor. Unfortunately, the Arduino's power supply proved insufficient to drive the sensor reliably. Our final solution relied on a software solution using approximate size calculations from the Pixy as a measure of distance. While this could not differentiate well between small and large objects, human and block-targeting

proved surprisingly accurate given their relatively constant sizes. (Just required adjusting sizing parameters.)

Limited Wire Length, Steering Considerations

In addition to our limited wire length, allowing 360+ rotations of the turret introduced steering challenges, primarily that the turret turned much faster than the car itself. This led us to limit the turret panning to a forward-facing 180 degrees; this provided the car time to react and, when combined with steering, had minimal effects on target following.

Poor Pixy2 Targeting

Due to its small package and low-quality camera, the Pixy had trouble identifying targets consistently. It failed to follow targets smoothly, and often made mistakes with sizing. This led our initial attempt at PID control to fail, as did subsequent attempts at filtering/averaging inputs. We eventually settled on a “block” based targeting system, where we had a center “block” inside which detection would not cause movement, while objects detected outside would cause movement in those directions at constant speeds (3 degrees per clock if far, 1 degree per clock if close). Despite the inevitable margin of error this introduced, the result proved to be surprisingly accurate and smooth.

7. Future Improvements (10 pts)

Suggest potential enhancements to your project.

Currently, the turret is equipped with only a laser. However, our original project proposal envisioned a more ambitious “gum launcher” feature, where the turret would aim and launch gum into a target’s mouth after the computer vision (CV) system tracked and identified it.

This concept was ultimately scrapped due to the extensive hardware design and integration required. To implement this in the future, we would need to develop a reliable projectile launching mechanism, incorporate more accurate distance sensing, and perform precise projectile motion calculations. While challenging, such an enhancement would significantly elevate the system’s functionality and showcase advanced targeting and tracking capabilities.

Our original design also relied on an OpenCV facial-recognition system using a much higher resolution webcam. However, this required a constant connection to a PC running a Python script, introducing unnecessary lag and the weight of a laptop. By sizing up the car or buying a mini-PC, this could be something we revert back to in the future.

8. Project Files (25 pts)

Submit all files in a ZIP folder with the following structure:

```
FinalProject_Groupxxx.zip
├── FinalReport.pdf
├── Arduino/
│   └── [design files]
├── RPi/
│   └── [design files]
└── PSoC/
    └── [design files]
```

> Files attached as instructed, all files also available at
>
> https://github.com/dandan002/laser-turret