Final Project Mechatronics

Nico Guerra, Peter Uba, Jay Heymann, Damian Cano

May 2024

1 Overall Design and Strategy

Our robot somewhat followed the proposed design and strategy as given in class and lecture. In practice, we used unique elements, like gripping belts and a solenoid, to help us achieve our desired outcome. While we recognized that simply pushing the puck into the opposing goal was an option, we decided to incorporate auxiliary mechanisms that would increase the likelihood of scoring and unique offensive maneuvers. The additional functionalities can be seen below:

Beam Breaks : We used a bream break sensor to detect the presence of the puck once within a certain threshold. Once at this threshold, the solenoid would be activated allowing the puck to be shot with some semblance of accuracy.

Belt Mechanism : We also incorporated a way to control the motion of the puck through the use of a belt mechanism. Connected to two motors on either side in a V formation, two belts would continuously rotate to drive the puck toward the middle of the robot, where the solenoid would then proceed to fire. The belts are also made of a light adhesive that would allow the puck to be controlled slightly easier.

Solenoid : Because we wanted to incorporate a shooting mechanism in our robot, we determined that using a solenoid would most likely produce the best outcome. Not only would it provide enough force to propel the puck, but it's a component straightforward to integrate into our robot's design.

1.1 Design Iteration

Initially, our robot had none of the above components. Instead, we decided that simply pushing the puck into the opposing goal would be enough. Quickly, we realized that a more robust method for scoring and puck control would be needed as well as an appropriate defense strategy. We scrapped our original ideas and added the aforementioned mechanical components. In terms of software, we saw a number of problems and necessary considerations after testing including: 1)

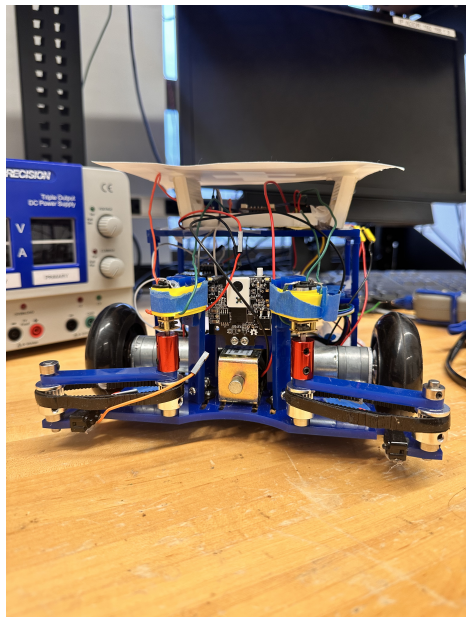


Figure 1: Final Robot Design
This is an image of our final robot design. The belts, solenoid, beam break sensors, and Pixy camera can be seen in this. Our wiring was slightly uncontained and wild but it worked.

locating the puck after its left the robot's field of vision and 2) appropriate PID control such that the puck can be directed into specific areas of the arena.

2 System Overview

2.1 Mechanical Design

The mechanical design of our final robot can be seen in **Figure 1**. The subsequent CAD can also be seen in **Figure 2** and **Figure 3**.

We opted for a simplistic design though in the end we did struggle for space to include each necessary component resulting in messy and cramped wiring and even using tape to hold pieces together. As can be seen in the CAD, our robot had two shelves upon which laid out each electrical component except for the pixy which had a separate mount. Having a simpler design allowed us to get started with testing faster rather than focusing on making the "best-looking" model.

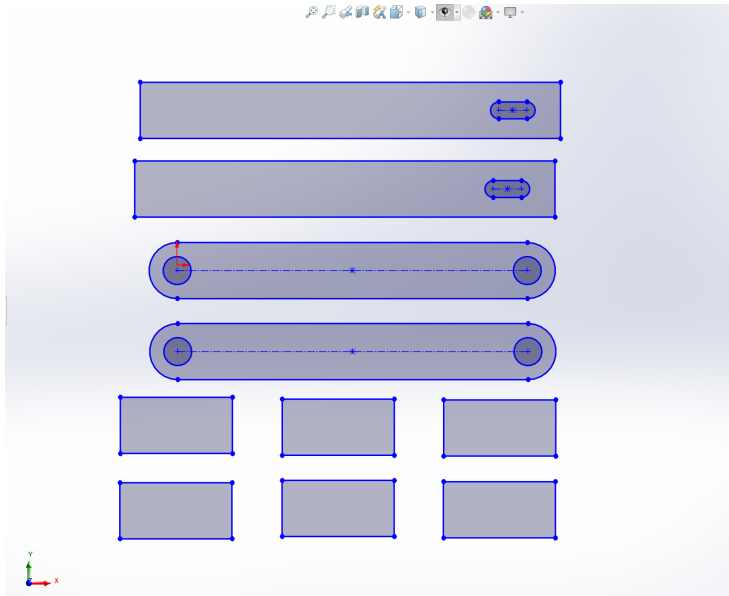


Figure 2: CAD Design 1
This CAD includes the structure on which the belt motors were mounted.

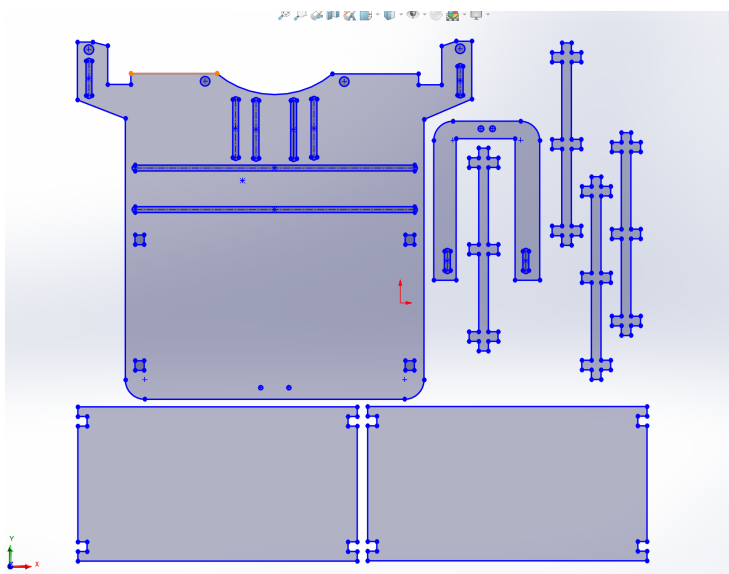
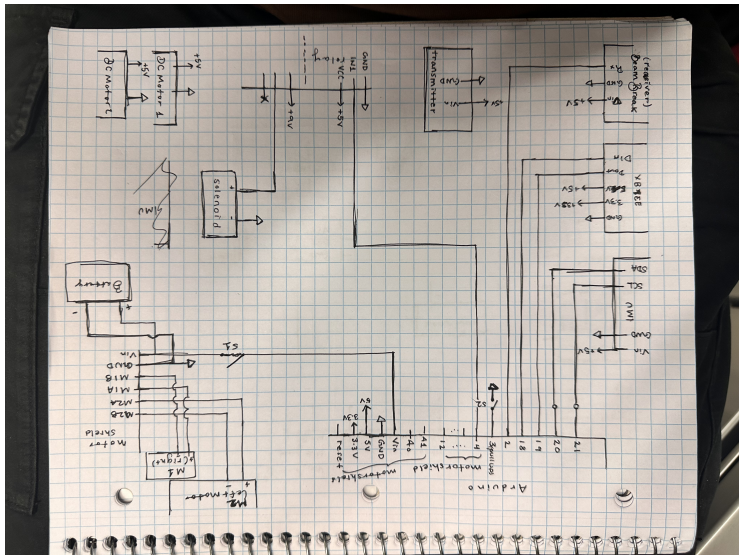


Figure 3: CAD Design 2
This CAD shows the chassis and the shelf structure.


Arduino
Motor Shield
Motor Shield
Motor Shield
Motor Shield
Battery
Solenoid
Transmitter
Transmitter
DC Motor
DC Motor
IMU
Pixy Camera
XBEE
Beam Break
Beam Break
Beam Break

7. Relay[1]: The relay was necessary to drive the solenoid and turn a small control signal from the Arduino into a large enough voltage to activate the solenoid.
8. Arduino: The primary microcontroller for the robot. It generates all of the control signals.
9. Solenoid[2]: The solenoid was used to shoot the puck once all necessary conditions were met.

2.3 Software

The software for our robot was well-written and organized utilizing the object-oriented programming capabilities of Arduino and C++. The classes utilized in the program are listed below:

MatchData Stored and parsed the match status and position data obtained from the ZigBee module.

PID Instance of a PID controller initialized with specified control parameters. Allowed easy creation and use of multiple simultaneous controllers for different applications, such as turning to a specified position or following the puck on the screen.

PixyVision Stored and parsed information from the Pixy camera to find the puck. Essential to allow the robot to chase the puck autonomously.

Movement The largest class, including both the overall state machine as well as the behavior for controlling the motors. Included separate behavior for following the puck continuously and moving to specified points on the arena using the coordinate system to determine position and the IMU to determine orientation.

In addition to those classes, multiple custom data structures were created to easily store and use the data needed by the various functions of the robot. These are listed below:

Target An object containing a desired position and orientation for the robot. When utilized by the Movement class, the robot will calculate the angle to rotate towards to face the target point, move to it, and then rotate to face the desired orientation.

ScreenPos An object containing the x and y position of an object on the screen. Utilized by the PixyVision class to calculate the horizontal error between the puck's position and the center of the screen to pass into a PID controller to adjust the speed of the motors, enabling precise tracking.

Outside of the classes, the program simply created instances of each object and called their corresponding setup and update methods when needed. As implemented in the Movement class, the state machine that governed the robot's overall logic is depicted in **Figure 5**.

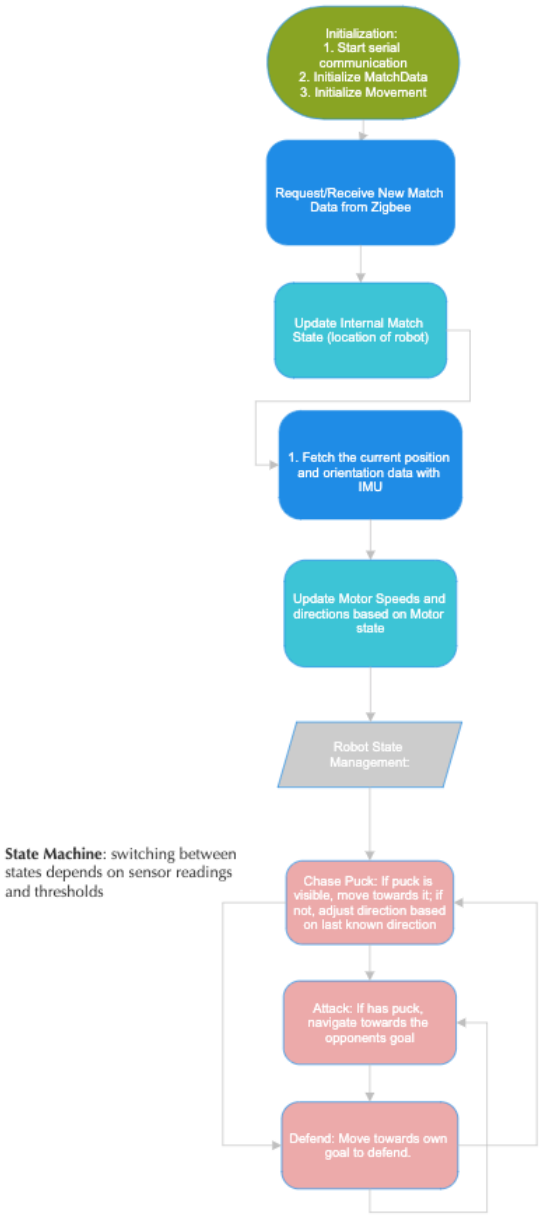


Figure 5: Control Scheme State Machine Diagram

Not depicted in the diagram is a separate nested state machine for the target system which split the behaviors of rotating towards the target point, moving to it, and rotating towards the target orientation. This was utilized in the Attack and Defend main states of the robot to move to pre-designated locations on the arena based on which team the robot was on.

Finally, during the final testing on match day an additional Unstuck state was added to the state machine which activated after no movement was detected for 4 seconds. When activated, the robot simply moved straight backwards for a short distance before re-engaging the state machine in the Chase Puck state. This helped rescue the robot when stuck to the walls and prevent stalemates when fighting for puck control, but was not implemented completely as desired due to the time constraints.

3 Testing

3.1 System Performance

The biggest positive for our robot's design was its consistency in certain situations. When the puck was in full view of the Pixy then our robot was able to move toward the puck and the appropriate goal in good time. The biggest problem that we ran into was incorporating an effective mechanism that would allow for the puck to be located when not in the Pixy's field of vision. In addition to this, we had to test consistently to determine the appropriate tuning that would allow for the puck to be controlled by the belts on our robot. Another problem we ran into was the adhesive belts sticking to the walls of the arena. This caused problems during our testing in that it required physical intervention.

Another problem we ran into was the adhesiveness of the belts. There was a sweet spot for belts in that they couldn't be too sticky else it'd count as grabbing; at the same time, the belts still needed some adhesive otherwise the whole system be rendered useless. To mitigate this problem, we considered evolving our game strategy to instead push the puck into the target goal. This worked decently except we lost a significant level of control in puck maneuvering.

A dramatic change that we employed was discarding the solenoid. After multiple iterations of testing, we decided to opt for simpler game logic since configuring the solenoid appropriately would've required a more complicated software design. In the end, this was a good decision because the simpler game logic allowed us to last longer in the competition.

3.2 Competition Day

4 Further Improvement

If there were time in the future, we would make a few improvements to our robot. For one, a better mechanism to control the puck would've helped the

overall performance of the robot. For example, instead of belts, using a built-in scooping "appendage" along with programming that would continuously swivel the robot in such a way that the puck is controlled and pushed into the desired location.

Another improvement that we would make is adding additional sensors to detect whenever the robot was approaching. This could be easily done using an ultrasonic sensor given its reliability in previous labs.

Aside from these important additions, there are other areas that could use improvement such as better wire management, a more spacious chassis/room for additional components, and even software that incorporates more defensive maneuvers to prevent the other team from scoring.

5 References

1.https://proteshea.com/control-air-valve-with-relay-and-arduino-uno/
2.https://www.adafruit.com/product/412
3.https://www.smartdraw.com/flowchart/flowchart-maker.htm
4.https://www.digikey.com/en/maker/tutorials/2022/add-an-ir-break-beam-sensor-to-an-arduino-project-for-object-detection

6 Appendix

6.1 Material List

Materials Provided / Found in Lab: - 2, 1/4 inch, square foot acrylic sheets - Zigbee module - Motor driver shield - Pixy Camera - 1 set of beam break sensors - Solenoid - 2 DC motors / encoders - 2 Motor mounting brackets - M3 screws and nuts - IMU - Solderless breadboard - Jumper wires - Painter's tape - Hot glue

Materials Purchased: - 1, 1/4 inch x square foot acrylic sheet - Timing belts and sprockets - Dowel pins - Ball bearings - Straight shaft couplings - Universal joint shaft couplings - Shaft Collars - M3 corner brackets - 2 high torque miniature DC motors - Double sided tape - Additional M3 screws and nuts

6.2 Budget

Item: 1, 1/4 inch x square foot acrylic sheet **Price:** 29.99 **Item:** Timing belts and sprockets **Price:** 14.99 **Item:** Dowel pin set **Price:** 13.99 **Item:** Ball bearing set **Price:** 9.59 **Item:** Straight shaft couplings set **Price:** 6.99 **Item:** Universal shaft couplings set **Price:** 11.18 **Item:** Shaft collars set **Price:** 8.99 **Item:** M3 corner brackets set **Price:** 6.99 **Item:** 2 High Torque Miniature DC

motors **Price:** 23.98 **Item:** 6 Miniature DC motors **Price:** 9.68 **Item:** Double Sided tape **Price:** 8.99 **Item:** M3 screws and nuts set **Price:** 9.39

Total Money Spent: 154.75 **Budget:** 200 **Money Remaining:** 45.25

6.3 Code

```
#include <Pixy2.h>
#include <DualMAX14870MotorShield.h>
#include "FireTimer.h"
#include <Wire.h>
#include <Adafruit_BNO055.h>
#include <math.h>

/*
* handles ZigBee communication
* also receives and stores current match data
*/
class MatchData
{
  bool status = 0; // true = match in progress
  int time = 0; // seconds
  int x = 0; // cm
  int y = 0; // cm

  bool isWaitingForUpdate = false;

  /*
  * read in the match data as a String from the ZigBee module
  * returns: String containing unparsed match data if available,
  *   empty String if data is not available
  */
  String receiveData()
  {
    String RX = ""; // format: M,TTTT,XXX,YYY

    int length = 0;
    bool isFinished = true;

    // first check if there's any data available at all before continuing
    if (Serial1.available())
    {
      isFinished = false;
    }
```

```

  // continue concatenating all data to the output string
  // THIS ASSUMES THE DATA IS FORMATTED AND BEING RECEIVED PROPERLY
  while (!isFinished)
  {
    if (Serial1.available())
    {
      char incoming = Serial1.read();
      ///Serial.println(incoming);
      RX.concat(incoming);
      length++;
    }

    if (length == 14)
    {
      isFinished = true;
    }
  }

  return RX;
}

void parseAndUpdate(String data)
{
  String str_status = data.substring(0, 1);
  String str_time = data.substring(2, 6);
  String str_x = data.substring(7, 10);
  String str_y = data.substring(11, 14);

  // if the new data isn't complete, fall back on previous data
  if (str_status.equals("?") || str_status.equals("/")
  || str_time.equals("????") || str_time.equals("////")
  || str_x.equals("---") || str_y.equals("---"))
  {
    return;
  }
  // new data is complete, go ahead with parsing

  bool isValid = true;

  bool status_new = str_status.equals("1");
  int time_new = str_time.toInt();
  int x_new = str_x.toInt();
  int y_new = str_y.toInt();

```

```
    // if the new data isn't valid for some reason, fall back on previous data
    // otherwise, go ahead and update data
    if (time_new < 0 || x_new < 0 || y_new < 0)
    {
      return;
    }

    status = status_new;
    time = time_new;
    x = x_new;
    y = y_new;
  }

public:
  bool getStatus() { return status; }
  int getTime() { return time; }
  int getX() { return x; }
  int getY() { return y; }

  void setup()
  {
    Serial1.begin(115200); // communicating w/ ZigBee (for match data)
  }

  void update()
  {
    // if the ZigBee hasn't been asked for an update yet, do that and return
    if (!isWaitingForUpdate)
    {
      Serial1.print('?');
      //Serial.println("requesting match update");
      isWaitingForUpdate = true;
      return;
    }
    // if we are waiting on the ZigBee module for an update, proceed

    // check on incoming data from ZigBee module
    String incoming = receiveData();
    // if data actually received, parse it and update
    if (!incoming.equals(""))
    {
      //Serial.print("received data: ");
```

```
      //Serial.println(incoming);
      parseAndUpdate(incoming);
      //print();
      // go back to requesting a new update
      isWaitingForUpdate = false;
    }
  }

  void print()
  {
    Serial.println("-----");

    Serial.print("Match in progress: ");
    Serial.println(status ? "YES" : "NO");

    Serial.print("Match time: ");
    Serial.print(time);
    Serial.println(" s");

    Serial.print("Robot position: (");
    Serial.print(x);
    Serial.print(", ");
    Serial.print(y);
    Serial.println(")");

    Serial.println("-----");
  }
};
MatchData match;

/*
* object that acts as an independent PID
* takes in values for the coefficients, a clamp for the output, and a clamp for the integral component
* the compute function takes in the input and setpoint, as well as updates the output directly by reference
*/
class PID
{
  const float Kp;
  const float Ki;
  const float Kd;

  const float iWindupMin;
  const float iWindupMax;
  //const float iErrorMin;
```

```
  //const float iErrorMax;

  const float outputMin;
  const float outputMax;

  const unsigned int samplePeriod_ms;

  float i = 0;

  float curError = 0;
  float lastError = 0;

  FireTimer timer;

  float doCompute(float input, float setpoint, ulong timeDiff)
  {
    lastError = curError;
    curError = setpoint - input;

    float p = Kp * curError;
    float d = Kd * (curError - lastError);
    float iTemp = 0;
    //if (curError >= iErrorMin && curError <= iErrorMax)
    //{
      iTemp = i + Ki * (curError + lastError) / 2.0;
    //}
    iTemp = constrain(iTemp, iWindupMin, iWindupMax);

    float out = p + d;
    /* part of code i didn't rly understand from the PID explanation i found -- leaving it out for now
    float iMax = constrain(outputMax - out, 0, outputMax);
    float iMin = constrain(outputMin - out, outputMin, 0);
    i = constrain(iTemp, iMin, iMax);
    */
    i = iTemp;

    out += i;
    return constrain(out, outputMin, outputMax);
  }

public:
  PID(float K_proportional, float K_integral, float K_derivative, float outputMin = -1000, float outputMax = 1000, float integralWindupMin = -1000, float integralWindupMax = 1000, unsigned int samplePeriod_ms = 50)
```

```
  : Kp(K_proportional), Ki(K_integral), Kd(K_derivative), outputMin(outputMin), outputMax(outputMax), iWindupMin(integralWindupMin), iWindupMax(integralWindupMax), samplePeriod_ms(samplePeriod_ms)
  {
    timer.begin(samplePeriod_ms);
  }

  /*
  // switched to an approach where we literally constrain the integral term to only be calculated within a certain error range rather than clamp the integral term
  // this is more like the approach my group took to lab 3/4 (? can't remember which one) so might be easier to tune for me
  PID(float K_proportional, float K_integral, float K_derivative, float outputMin = -1000, float outputMax = 1000, float integralErrorMin = -1000, float integralErrorMax = 1000, unsigned int samplePeriod_ms = 50)
  : Kp(K_proportional), Ki(K_integral), Kd(K_derivative), outputMin(outputMin), outputMax(outputMax), iErrorMin(integralErrorMin), iErrorMax(integralErrorMax), samplePeriod_ms(samplePeriod_ms)
  {
    timer.begin(samplePeriod_ms);
  }
  */

  void start()
  {
    reset();

    timer.start();
  }

  void reset()
  {
    curError = 0;
    lastError = 0;
    i = 0;
  }

  void compute(float input, float setpoint, float &output)
  {
    if (timer.fire())
    {
```

```
      output = doCompute(input, setpoint, timer.timeDiff);
    }
  }
};

struct ScreenPos
{
  int x;
  int y;

  ScreenPos(int x = -1, int y = -1) : x(x), y(y) { }

  ScreenPos& operator=(const ScreenPos& other)
  {
    x = other.x;
    y = other.y;
    return *this;
  }

  bool operator==(const ScreenPos& other) const
  {
    return (x == other.x && y == other.y);
  }
};
const ScreenPos SCREEN_POS_CENTER = ScreenPos(315/2, 207/2);
const ScreenPos SCREEN_POS_NULL = ScreenPos(-1, -1);

class PixyVision
{
  Pixy2 pixy;

  // number = bitmap to select for that signature (1 byte, 128-place specifies color codes)
  enum PixySignature
  {
    PUCK = 1, // orange,
    GOAL_RED = 2, // red
    GOAL_PINK = 4 // pink
  };

  unsigned long puckScreenTime = 0;
  int lastDir = 0; // -1 for left, 1 for right
  const unsigned long PUCK_FIND_DURATION = 1000; // ms to look towards the side of the screen the puck disappeared from before giving up

```

```
  float speedAdjust = 0;

  ScreenPos pixyScan(PixySignature signatureBitmap, int maxBlocks)
  {
    /*
    false -> don't wait for block data to continue
    PixySignature -> specifies which signature to select
    int -> max blocks to select (sorted by size)
    */
    uint8_t numBlocks = pixy.ccc.getBlocks(false, signatureBitmap, maxBlocks);

    // didn't find any objects, break out of the method immediately
    if (pixy.ccc.blocks <= 0)
    {
      return SCREEN_POS_NULL;
    }

    int xsum = 0;
    int ysum = 0;
    for (uint8_t i = 0; i < numBlocks; i++)
    {
      Block curBlock = pixy.ccc.blocks[i];
      xsum += curBlock.m_x;
      ysum += curBlock.m_y;
    }

    return ScreenPos(xsum / numBlocks, ysum / numBlocks);
  }

public:
  void setup()
  {
    pixy.init();
  }

  ScreenPos scanPuck()
  {
    ScreenPos screenPosPuck = pixyScan(PixySignature::PUCK, 3);
    if (!(screenPosPuck == SCREEN_POS_NULL))
    {
      lastDir = (screenPosPuck.x - SCREEN_POS_CENTER.x < 0) ? 1 : -1;
      puckScreenTime = millis();
```

```
  }

  return screenPosPuck;
}

/*
* -1 = off the left side of the screen
* 0 = been too long since it disappeared from the screen
* 1 = off the right side of the screen
*/
int findPuck()
{
  unsigned long curTime = millis();
  if (curTime - puckScreenTime <= PUCK_FIND_DURATION)
  {
    Serial.print("puck screen timer: ");
    Serial.println(curTime - puckScreenTime);
    return lastDir;
  }

  return 0;
}

ScreenPos scanRedGoal()
{

  ScreenPos screenPosGoalRed =
  ↪  pixyScan(PixySignature::GOAL_RED, 1);
  return screenPosGoalRed;

  /*
  Serial.print("Red Goal Screen Pos: (");
  Serial.print(screenPosGoalRed.x);
  Serial.print(", ");
  Serial.print(screenPosGoalRed.y);
  Serial.print(") => error: ");
  if (screenPosGoalRed == SCREEN_POS_NULL)
  {
    Serial.println("not on screen");
  }
  else
  {
    Serial.println(SCREEN_POS_CENTER.x - screenPosGoalRed.x);
  }
  */
}
```

```
  ScreenPos scanPinkGoal()
  {
    ScreenPos screenPosGoalPink = pixyScan(PixySignature::GOAL_PINK, 1);
    return screenPosGoalPink;

    /*
    Serial.print("Pink Goal Screen Pos: (");
    Serial.print(screenPosGoalPink.x);
    Serial.print(", ");
    Serial.print(screenPosGoalPink.y);
    Serial.print(") => error: ");
    if (screenPosGoalPink == SCREEN_POS_NULL)
    {
      Serial.println("not on screen");
    }
    else
    {
      Serial.println(SCREEN_POS_CENTER.x - screenPosGoalPink.x);
    }
    Serial.println("");
    */
  }
};

struct Target
{
  int x;
  int y;
  float angle;

  Target(int x, int y, float angle) : x(x), y(y), angle(angle) { }
};

//const Target TG_ATTACK_CENTER_RED = Target(40, 60, 180);
const Target TG_ATTACK_CENTER_RED = Target(25, 60, 180);
const Target TG_ATTACK_LEFT_RED = Target(35, 35, 135);
const Target TG_ATTACK_RIGHT_RED = Target(35, 90, 225);

//const Target TG_ATTACK_CENTER_PINK = Target(195, 60, 0);
const Target TG_ATTACK_CENTER_PINK = Target(210, 60, 0);
const Target TG_ATTACK_LEFT_PINK = Target(200, 90, 315);
const Target TG_ATTACK_RIGHT_PINK = Target(200, 35, 45);
```

```
const Target TG_DEFEND_PINK = Target(205, 60, 180);
const Target TG_DEFEND_RED = Target(35, 60, 0);

/*
* stores position and orientation data
* these will be computed relatively based on match data (for position) and the IMU (for orientation)
* this will also handle determining which zone the robot is in, as well as moving to a target position
*/
class Movement
{
  const static float RAD2DEG = 180 / 3.14159;

  const int MOTOR_SPEED_F = 150; // range from -400 to 400

  const unsigned long STOP_PERIOD = 400; // MS to wait between hitting target position and swinging to target angle
  unsigned long stopTime = 0;

  Adafruit_BNO055 bno = Adafruit_BNO055(55, 0x28, &Wire);
  DualMAX14870MotorShield motors;
  PixyVision pixyVision;

  const int PIN_BEAMBREAK = 2;
  const int PIN_TEAM = 3;

  enum {
    RED_TEAM, // defending red goal (closer to origin), attacking pink
    PINK_TEAM // defending pink goal, attacking red goal
  } team;

  enum {
    GAME_OVER,
    CHASE_PUCK,
    ATTACK,
    DEFEND,
    UNSTUCK
  } robotState;

  bool _hasPuck = false;

  float baseAngle = 0;

  float angle = 0;
```

```
int x = 0;
int y = 0;

enum {
  NONE,
  TARGET_SET,
  MOVE_TO_POS,
  STOP,
  MOVE_TO_ANGLE
} targetState;
int targetX = 0;
int targetY = 0;
float targetAngle = 0;
float targetAngleIntermediate = 0;

float turnAdjust = 0;
float speedAdjust = 0;
float trackAdjust = 0;

//PID pid_adjustTurning = PID(2.4, 3.5, 0.01, -400, 400);
PID pid_adjustTurning = PID(10, 0, 20, -400, 400);
PID pid_pureTurning = PID(10, 10, 20, -200, 200, -75, 75);
//PID pid_pureTurning = PID(10, 10, 20, -400, 400, -50, 50);
PID pid_movement = PID(5, 0, 5, 0, MOTOR_SPEED_F);
PID pid_puckTracker = PID(2, 0, 4, -200, 200, -75, 75);

unsigned long abeTimePos = 0;
unsigned long abePrevTimePos = 0;
const unsigned long ABOUTEQUALS_TIMER_POSITION = 100; // time in MS position should be within the threshold before returning true
const float ABOUTEQUALS_RANGE_POSITION = pow(4, 2); // square distance in CM(^2)
bool aboutEqualsPosition(float testX, float testY, float targetX, float targetY)
{
  // check if in range using square distance (sqrt is slow, idk how relevant that is tho for this application tbh)
  bool isInRange = squareDistanceBetween(testX, testY, targetX, targetY) <= ABOUTEQUALS_RANGE_POSITION;

  unsigned long curTime = millis();
  if (isInRange) // still within range, just check how long it's been
  {
    abeTimePos += curTime - abePrevTimePos;
```

```
      if (abeTimePos >= ABOUTEQUALS_TIMER_POSITION) // sufficient time within range
      {
        abeTimePos = 0;
        return true;
      }
    }
    else // out of range now, reset the timer
    {
      abeTimePos = 0;
    }

    abePrevTimePos = curTime;
    return false;
  }

  unsigned long abeTimeAngle = 0;
  unsigned long abePrevTimeAngle = 0;
  const unsigned long ABOUTEQUALS_TIMER_ANGLE = 100; // time in MS angle should be within the threshold before returning true
  const float ABOUTEQUALS_RANGE_ANGLE = 1.5; // angle in degrees -> [target - range, target + range]
  bool aboutEqualsAngle(float testAngle, float targetAngle)
  {
    // check if in range using square distance (sqrt is slow, idk how relevant that is tho for this application tbh)
    bool isInRange = abs(targetAngle - testAngle) <= ABOUTEQUALS_RANGE_ANGLE;

    unsigned long curTime = millis();
    if (isInRange) // still within range, just check how long it's been
    {
      abeTimeAngle += curTime - abePrevTimeAngle;

      if (abeTimeAngle >= ABOUTEQUALS_TIMER_ANGLE) // sufficient time within range
      {
        abeTimeAngle = 0;
        return true;
      }
    }
    else // out of range now, reset the timer
    {
```

```
      abeTimeAngle = 0;
    }

    abePrevTimeAngle = curTime;
    return false;
  }

  float angleError(float input, float setpoint)
  {
    float e = setpoint - input;
    if (e > 180)
    {
      e -= 360;
    }
    else if (e < -180)
    {
      e += 360;
    }

    return e;
  }

  float angleBetween(float x1, float y1, float x2, float y2)
  {
    return wrapAngle(atan2f(y2 - y1, x2 - x1) * RAD2DEG);
  }

  float wrapAngle(float input)
  {
    return input - 360 * floor(input/360);
  }

  float squareDistanceBetween(float x1, float y1, float x2, float
  ↪  y2)
  {
    return pow(y2 - y1, 2) + pow(x2 - x1, 2);
  }

  bool trackPuck()
  {
    ScreenPos screenPosPuck = pixyVision.scanPuck();

    Serial.print("Puck Screen Pos: (");
    Serial.print(screenPosPuck.x);
    Serial.print(", ");
    Serial.print(screenPosPuck.y);
```

```
    Serial.print(")");
    float s;
    bool b = true;
    if (screenPosPuck == SCREEN_POS_NULL)
    {
      int lastDir = pixyVision.findPuck();
      Serial.print("lastDir: ");
      Serial.println(lastDir);
      if (lastDir == 0)
      {
        s = 0;
        trackAdjust = 0;
        Serial.println(" => not on screen -- stop searching");
        b = false;
      }
      else
      {
        s = 0;
        trackAdjust = lastDir * 150;
        Serial.print(" => not on screen -- still looking => ");
        Serial.println(trackAdjust);
      }
    }
    else
    {
      s = MOTOR_SPEED_F;
      pid_puckTracker.compute(screenPosPuck.x, SCREEN_POS_CENTER.x, trackAdjust);
      Serial.print(" => ");
      Serial.println(trackAdjust);
    }

    setMotorSpeed(s - trackAdjust, s + trackAdjust);
    return b;
  }

public:
  void setup()
  {
    if (!bno.begin())
    {
      Serial.println("BNO055 IMU not detected!");
      while (1);
    }

    pixyVision.setup();
```

```
  motors.enableDrivers();
  motors.flipM2(true);

  pinMode(PIN_BEAMBREAK, INPUT_PULLUP);
  pinMode(PIN_TEAM, INPUT_PULLUP);

  pid_adjustTurning.start();
  pid_pureTurning.start();
  pid_movement.start();
  pid_puckTracker.start();

  int t = digitalRead(PIN_TEAM);
  if (t == LOW)
  {
    team = RED_TEAM;
  }
  else
  {
    team = PINK_TEAM;
  }
  Serial.print("TEAM: ");
  Serial.println(team == RED_TEAM ? "RED" : "PINK");
  robotState = CHASE_PUCK;

  sensors_event_t orientationData;
  bno.getEvent(&orientationData,
  ↪  Adafruit_BNO055::VECTOR_EULER);

  baseAngle = orientationData.orientation.x;

  unsigned long curTime = millis();
  abeTimePos = curTime;
  abeTimeAngle = curTime;
}

void print()
{
  Serial.print("MOVEMENT: (");
  Serial.print(x);
  Serial.print(", ");
  Serial.print(y);
  Serial.print(") | ");
  Serial.println(angle);
}
```

```
void setMotorSpeed(float leftSpeed, float rightSpeed)
{
  motors.setM1Speed(rightSpeed);
  motors.setM2Speed(leftSpeed);
}

unsigned long defendTimer = 0;
const unsigned long DEFEND_WAIT_DURATION = 2000; // ms to wait for puck to reappear before going back to defending
unsigned long puckGrabTimer = 0;
const unsigned long PUCK_GRAB_DURATION = 500; // ms to wait after losing puck before switching to chase mode
unsigned long unstuckMoveTimer = 0;
const unsigned long UNSTUCK_MOVE_DURATION = 1000; // ms to move backwards after getting stuck
unsigned long unstuckActivateTimer = 0;
const unsigned long UNSTUCK_ACTIVATE_DURATION = 3000; // ms to wait when stuck before moving backwards

int lastX = 0;
int lastY = 0;

void update(int x, int y)
{
  this->x = x;
  this->y = y;

  sensors_event_t orientationData;
  bno.getEvent(&orientationData, Adafruit_BNO055::VECTOR_EULER);

  // offset the angle by the base angle and wrap to [0, 360)
  angle = wrapAngle(-orientationData.orientation.x + baseAngle);

  unsigned long curTime = millis();

  int bb = digitalRead(PIN_BEAMBREAK);
  if (bb == LOW)
  {
    _hasPuck = true;
    puckGrabTimer = curTime;
  }
  else
  {
    _hasPuck = false;
```

```
    }


    if (targetState == NONE && robotState != UNSTUCK)
    {
      if (!aboutEqualsPosition(lastX, lastY, x, y))
      {
        lastX = x;
        lastY = y;
        unstuckActivateTimer = curTime;
      }
    }

    if (curTime - unstuckActivateTimer >= UNSTUCK_ACTIVATE_DURATION && robotState != UNSTUCK)
    {
      resetTarget();
      robotState = UNSTUCK;
      unstuckMoveTimer = curTime;
    }


    if (targetState == NONE)
    {
      //Serial.println("---- NO TARGET");
      //trackPuck();
    }
    else
    {
      Serial.print("==== TARGET: (");
      Serial.print(targetX);
      Serial.print(", ");
      Serial.print(targetY);
      Serial.print(") | ");
      Serial.print(targetAngleIntermediate);
      Serial.print(", ");
      Serial.println(targetAngle);
    }

    switch (targetState)
    {
      // initial target has been set:
      // move to the orientation required to go forward to the correct position
      case TARGET_SET:
      {
```

```
        Serial.println("TARGET SET:");
        pid_pureTurning.compute(-angleError(angle, targetAngleIntermediate), 0, turnAdjust);
        setMotorSpeed(0 - turnAdjust, 0 + turnAdjust);

        Serial.print("\tangle: ");
        Serial.print(angle);
        Serial.print(" | target: ");
        Serial.print(targetAngleIntermediate);
        Serial.print(" | error: ");
        Serial.println(angleError(angle, targetAngleIntermediate));
        Serial.print("\tturnAdjust: ");
        Serial.println(turnAdjust);

        if (aboutEqualsAngle(angle, targetAngleIntermediate))
        {
          targetState = MOVE_TO_POS;
          pid_adjustTurning.reset();
          pid_movement.reset();
        }
        break;
      }

      // we're facing the correct position now:
      // move to the correct position
      case MOVE_TO_POS:
      {
        Serial.println("MOVE TO POS:");

        targetAngleIntermediate = angleBetween(x, y, targetX, targetY);
        float squareDist = squareDistanceBetween(x, y, targetX, targetY);

        pid_adjustTurning.compute(-angleError(angle, targetAngleIntermediate), 0, turnAdjust);
        pid_movement.compute(-squareDist, 0, speedAdjust);
        setMotorSpeed(speedAdjust - turnAdjust, speedAdjust + turnAdjust);

        Serial.print("\tangle: ");
        Serial.print(angle);
        Serial.print(" | target: ");
        Serial.print(targetAngleIntermediate);
        Serial.print(" | error: ");
```

```
        Serial.println(angleError(angle, targetAngleIntermediate));
        Serial.print("\tspeedAdjust: ");
        Serial.println(turnAdjust);
        Serial.print("\tposition: (");
        Serial.print(x);
        Serial.print(", ");
        Serial.print(y);
        Serial.print(") | target: (");
        Serial.print(targetX);
        Serial.print(", ");
        Serial.print(targetY);
        Serial.print(") | square distance: ");
        Serial.println(squareDist);

        if (aboutEqualsPosition(x, y, targetX, targetY))
        {
          stopTime = millis();
          targetState = STOP;
        }

        /*
        unsigned long curTime = millis();
        if (curTime - tempTimer >= 2000) // wait time since we don't have position checking up yet
        {
          stopTime = curTime;
          targetState = NONE;
        }
        */

        break;
      }

      case STOP:
      {
        Serial.println("BRIEF PAUSE TO SETTLE");
        setMotorSpeed(0, 0);

        if (curTime - stopTime >= STOP_PERIOD)
        {
          // no target angle set, we're done here
          if (targetAngle < 0)
          {
            targetState = NONE;
            turnAdjust = 0;
```

```
            setMotorSpeed(0, 0);
          }
          else
          {
            targetState = MOVE_TO_ANGLE;
            pid_pureTurning.reset();
          }
        }
        break;
      }

      // we're in the correct position now:
      // move to the correct orientation (if there is a target angle set)
      case MOVE_TO_ANGLE:
      {
        Serial.println("MOVE TO ANGLE:");
        pid_pureTurning.compute(-angleError(angle, targetAngle), 0, turnAdjust);
        setMotorSpeed(0 - turnAdjust, 0 + turnAdjust);

        Serial.print("\tangle: ");
        Serial.print(angle);
        Serial.print(" | target: ");
        Serial.print(targetAngle);
        Serial.print(" | error: ");
        Serial.println(angleError(angle, targetAngle));

        if (aboutEqualsAngle(angle, targetAngle))
        {
          targetState = NONE;
          turnAdjust = 0;
          setMotorSpeed(0, 0);
        }
        break;
      }

      // else (case NONE) do nothing
      default:
      {
        break;
      }
    }

    switch (robotState)
    {
```

```
        case CHASE_PUCK:
        {
          Serial.println("----- CHASE PUCK -----");
          bool track = trackPuck();

          if (!track) // lost track of puck for too long
          {
            setTarget(team == RED_TEAM ? TG_DEFEND_RED : TG_DEFEND_PINK);
            robotState = DEFEND;
          }

          if (_hasPuck)
          {
            setTarget(team == RED_TEAM ? TG_ATTACK_CENTER_PINK : TG_ATTACK_CENTER_RED);
            robotState = ATTACK;
          }
          break;
        }

        case ATTACK:
        {
          Serial.println("----- ATTACK -----");
          if (curTime - puckGrabTimer >= PUCK_GRAB_DURATION)
          {
            resetTarget();
            robotState = CHASE_PUCK;
          }
          break;
        }

        case DEFEND:
        {
          Serial.println("----- DEFEND -----");
          if (targetState == NONE) // has reached the defend position)
          {
            robotState = CHASE_PUCK;
          }
          break;
        }

        case UNSTUCK:
        {
          Serial.println("----- UNSTUCK -----");
```

```
        Serial.print("cur time: ");
        Serial.print(curTime);
        Serial.print(" | move timer: ");
        Serial.print(unstuckMoveTimer);
        Serial.print(" | duration: ");
        Serial.println(curTime - unstuckMoveTimer);
        setMotorSpeed(-150, -150);
        if (curTime - unstuckMoveTimer >= UNSTUCK_MOVE_DURATION)
        {
          robotState = CHASE_PUCK;
          unstuckActivateTimer = curTime;
        }
        break;
      }

      default: // GAME_OVER
      {
        break;
      }
    }


    /* CODE ONLY USED FOR TURNING PID TEST
    Serial.println("TARGET SET:");
    pid_pureTurning.compute(-angleError(angle, targetAngle), 0, speedAdjust);
    setMotorSpeed(0 - speedAdjust, 0 + speedAdjust);
    Serial.print("\tangle: ");
    Serial.print(angle);
    Serial.print(" | target: ");
    Serial.print(targetAngle);
    Serial.print(" | error: ");
    Serial.println(angleError(angle, targetAngle));
    Serial.print("\tspeedAdjust: ");
    Serial.println(speedAdjust);
    //*/

    /* CODE ONLY USED FOR MOVING PID TEST
    Serial.println("FORWARD MOVEMENT SET:");
    targetAngle = 0;
    pid_adjustMovement.compute(-angleError(angle, targetAngle), 0, speedAdjust);
    setMotorSpeed(MOTOR_SPEED_F - speedAdjust, MOTOR_SPEED_F + speedAdjust);
    Serial.print("\tangle: ");
    Serial.print(angle);
```

```
    Serial.print(" | target: ");
    Serial.print(targetAngle);
    Serial.print(" | error: ");
    Serial.println(angleError(angle, targetAngle));
    Serial.print("\tspeedAdjust: ");
    Serial.println(speedAdjust);
    //*/
  }

  /*
  void setTargetAngle(int targetAngle)
  {
    this->targetAngle = targetAngle;
  }
  */

  bool hasPuck()
  {
    return _hasPuck;
  }

  void setTarget(Target target)
  {
    targetState = TARGET_SET;
    targetX = target.x;
    targetY = target.y;
    targetAngle = target.angle;

    turnAdjust = 0;
    setMotorSpeed(0, 0);

    if (aboutEqualsPosition(x, y, targetX, targetY))
    {
      // no target angle set, we're done here
      if (targetAngle < 0)
      {
        targetState = NONE;
      }
      else
      {
        targetState = MOVE_TO_ANGLE;
        pid_pureTurning.reset();
      }
    }

    /*
```

```
    //float t = atan2f(x - targetX, targetY - y) * RAD2DEG;
    float t = atan2f(targetY - y, targetX - x) * RAD2DEG;
    targetAngleIntermediate = wrapAngle(t);
    */
    targetAngleIntermediate = angleBetween(x, y, targetX,
    ↪  targetY);
    pid_pureTurning.reset();
  }

  void resetTarget()
  {
    targetState = NONE;
    setMotorSpeed(0, 0);
  }

  bool isTargeting()
  {
    return targetState != NONE;
  }
};
Movement movement;

unsigned long waitTimer = 0;
const unsigned long TIME_TO_WAIT = 1000;
void setup()
{
  Serial.begin(115200); // console output

  delay(2000);

  match.setup();
  movement.setup();

  //movement.setTarget(10, 10, 45);
  //movement.setTargetAngle(45);
  waitTimer = millis();

  //movement.setMotorSpeed(150, 150);
}

/*
int i = -1;
bool b = true;
Target targets[] = {
  Target(35, 35, 45), // red right corner
  Target(35, 90, 315), // red left corner
```

```
  Target(205, 90, 225), // pink right corner
  Target(205, 35, 135), // pink left corner
  Target(120, 60, 0), // center, facing pink
  Target(120, 60, 180), // center, facing red

  stateMachine.TG_ATTACK_CENTER_RED,
  stateMachine.TG_ATTACK_LEFT_RED,
  stateMachine.TG_ATTACK_RIGHT_RED,
  stateMachine.TG_ATTACK_CENTER_PINK,
  stateMachine.TG_ATTACK_LEFT_PINK,
  stateMachine.TG_ATTACK_RIGHT_PINK,
  stateMachine.TG_DEFEND_RED,
  stateMachine.TG_DEFEND_PINK,
  stateMachine.TG_DEFEND_RED
};
int targets_len = 15;
*/

void loop()
{
  // fetch latest match data from ZigBee
  match.update();

  /*
  if (!match.getStatus())
  {
    Serial.println("***** MATCH STOPPED *****");
    movement.resetTarget();
    movement.setMotorSpeed(0, 0);
    return;
  }
  */

  //match.print();
  movement.update(match.getX(), match.getY());
  //movement.update(60, 60);
  //delay(1000);

  if (millis() - waitTimer < TIME_TO_WAIT)
  {
    return;
  }

  //match.print();
  //movement.print();

```

```
  /*
  if (b)
  {
    if (!movement.isTargeting())
    {
      b = false;
      i++;
    }
  }

  if (!b && i < targets_len)
  {
    movement.setTarget(targets[i]);
    b = true;
  }
  */

  delay(50);
}
```