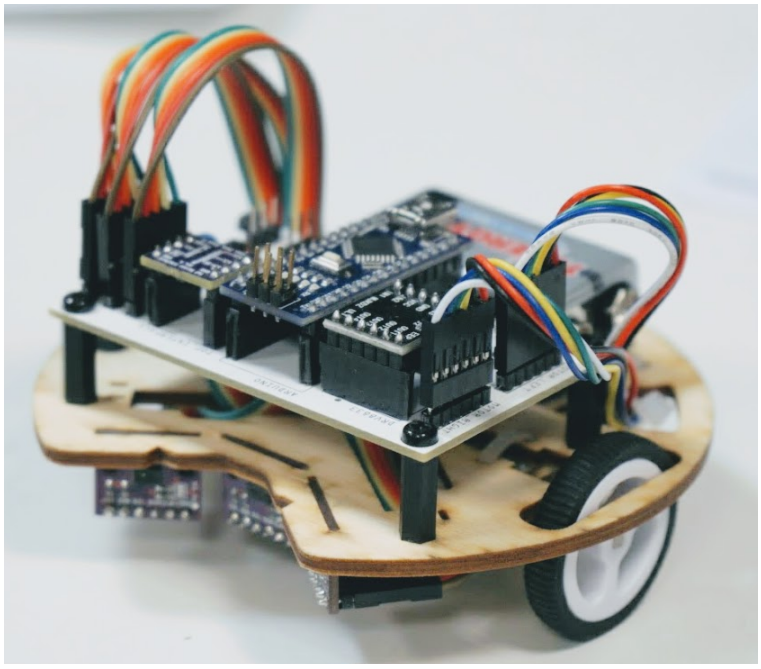
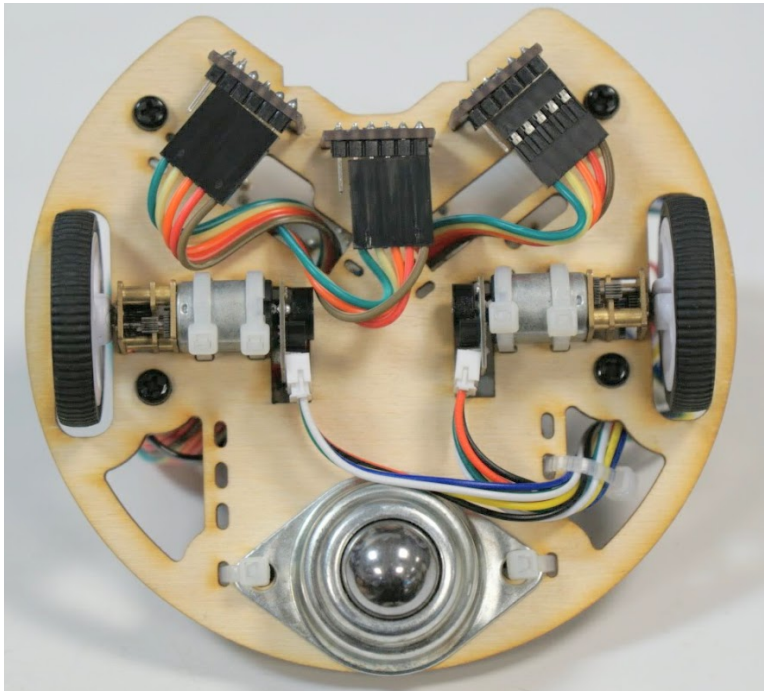
Lab 4 - Build Day and Odometry

- Robot Assembly
- Odometry
    - Converting to Real Units
    - Velocity Measurement
    - Differential Drive Robot Odometry
    - Arduino Code

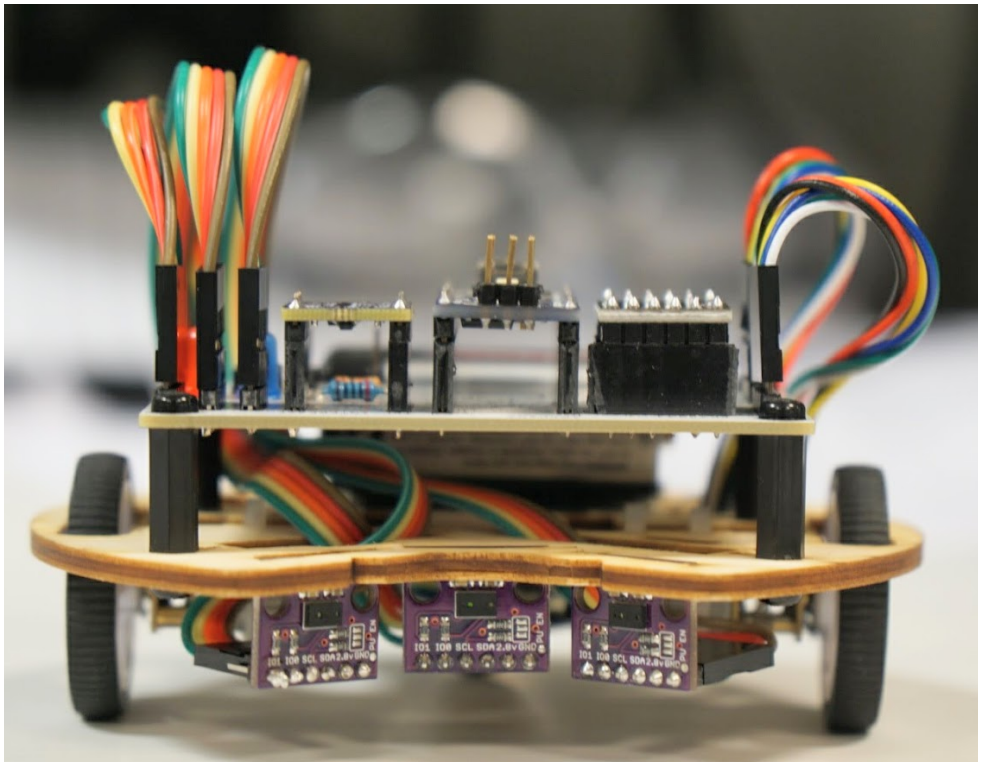
Robot Chassis Assembly

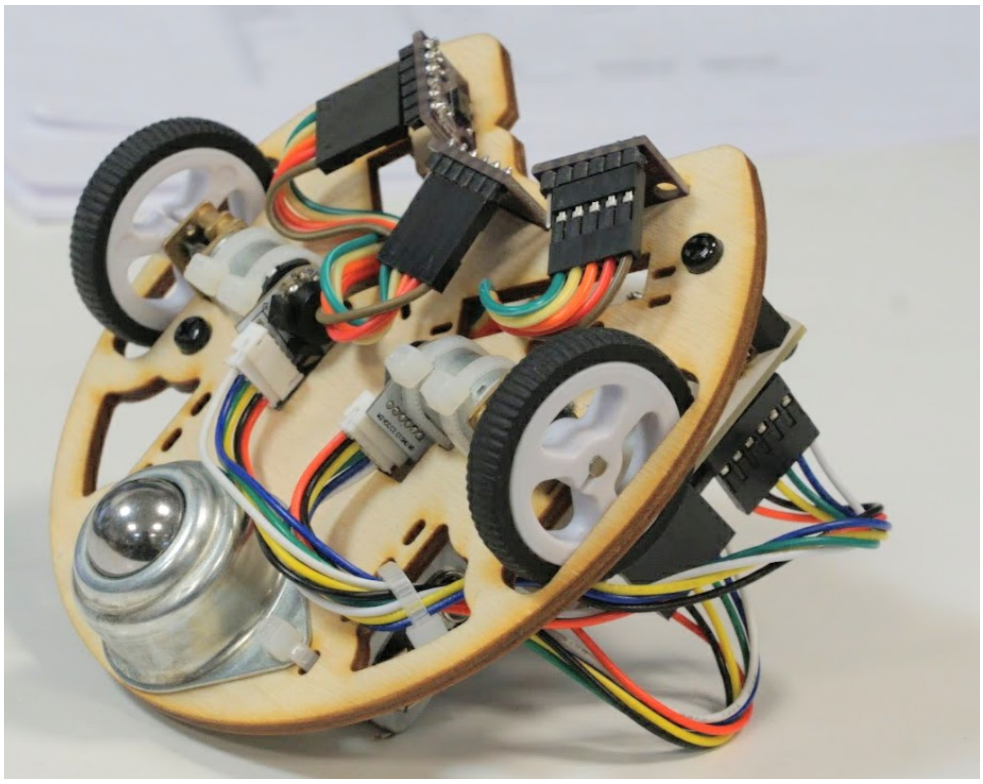
The first thing we'll be doing today is assembling the chassis of your micromouse! This is what it should look like at the end:





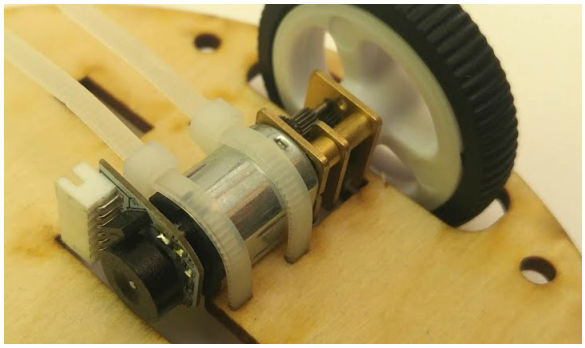
LAB 04





A couple notes to keep in mind while you're doing this:

- **ToF sensors:** These do require some force to press into the chassis, but still be careful not to break them! Ask if you're unsure.
- **Motors:** these are also attached with zip ties. Again, watch the orientation! The connectors should face horizontally out the back of the robot.



Checkoff #1

1. Show your mentor your assembled chassis!

Odometry

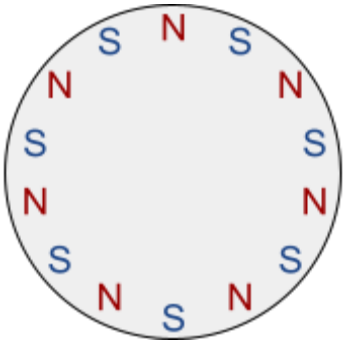
Last week, we learned how to read the encoders on our mouse. Encoders are a type of sensor that tells us how many times each wheel rotated. Using this information, we can calculate how far our mouse travelled and how many degrees it turned. This is a form of *odometry*, the use of data from motion sensors to estimate change in position over time[1].

Why is odometry important? Suppose we want our mouse to move forward one maze square. We could turn on both motors for a fixed amount of time, and hope the mouse moves exactly the distance we want it to. In practice, this doesn't work due to friction, changing battery voltage, and many other factors. The difference between the desired movement and the actual movement is called *error*, and knowing the error is the first step towards correcting it.

Converting to Real Units

Earlier, you learned how to count the number of encoder ticks for each wheel. Right now, these wheel position values don't mean much. It's generally a good idea to do calculations in real units (millimeters, radians, etc.) whenever possible. How do we convert encoder ticks to actual distance travelled, in millimeters?

Let's start by finding the number of motor rotations per encoder tick. Recall last week's description of how the encoders on our mouse work. A magnet is attached to the shaft of the motor, and two Hall effect sensors output a signal depending on whether the north or south pole of the magnet is facing them. There is one detail we didn't mention: our magnet wheel has seven north poles and seven south poles, so the Hall effect sensors output seven pulses per motor revolution. In other words, the motor spins one seventh of a revolution per encoder tick.



Encoder wheel

You may have noticed that the wheels on our mouse spin much slower than the motors. That's because the wheel is connected to the motor through a *gearbox*, which reduces speed while

[1] Thanks, Wikipedia

increasing torque. Most motors in robots need a gearbox to produce useful amounts of torque. The gearboxes on our motors have a 30:1 ratio, meaning that the wheel spins once for every 30 motor revolutions.

Finally, we need to know how much distance (in millimeters) a wheel covers per revolution. This depends on the diameter of the wheel. Our wheels have a diameter of 34 mm.

Using these facts, we can derive a conversion factor that goes from encoder ticks to distance travelled by a wheel (in millimeters).

Question: How many millimeters does a wheel travel per encoder tick?

Velocity Measurement

Knowing how fast each wheel on our mouse is moving would be really useful. As long as both wheels are moving at the same velocity, our mouse will drive straight. How do we measure velocity from encoder ticks?

Velocity is defined as distance traveled divided by time. Note that we are measuring distance using encoders, which can only output a whole number of ticks. This leaves us with two ways to compute velocity. We can count the number of encoder ticks during a fixed period of time, or measure the amount of time per encoder tick.

The second method has much higher precision, because time can be measured in microseconds[2] while encoder ticks can only take on a few different values. We will use the second method. Every time we get a rising edge interrupt from an encoder, we measure the number of microseconds elapsed since the last rising edge. Dividing the distance of one encoder tick by the time elapsed gives us the current velocity.

The Arduino code below computes wheel velocities using this method. Note that there is an edge case, which is handled in `checkEncodersZeroVelocity()`.

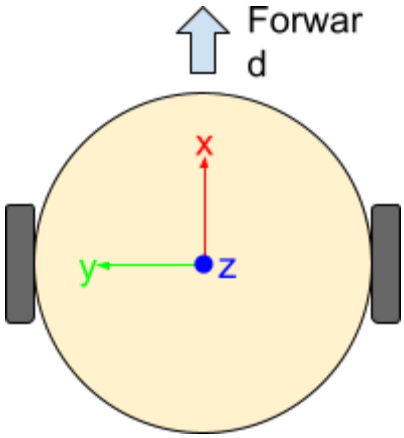
Question(s): The `checkEncodersZeroVelocity()` function sets wheel velocities to zero if it hasn't seen an encoder pulse in 100 milliseconds. Why do we need this? Will the interrupt callbacks (`leftEncoderRisingEdge` or `rightEncoderRisingEdge`) ever set a wheel velocity to 0?

[2] Using the Arduino function `micros()`

Differential Drive Robot Odometry

Now that we have odometry information (distance and velocity) for each wheel, let's figure out how to compute odometry for the entire mouse. Our mouse is a *differential drive* robot, which means that it has two drive wheels located side by side. Both wheels turning in the same direction causes forward (or backward) motion, and both wheels turning in opposite directions causes rotation.

Sidenote
When we describe the position and velocity of a robot, we are actually describing the position and velocity of a single point somewhere on the robot. For a differential drive robot like our mouse, this point is usually taken to be halfway between the wheels. Since a point can't have a rotational velocity, the "point" is actually a *coordinate frame*. You can think of a coordinate frame as a set of coordinate axes that are "attached" to the mouse and move/rotate with it.



Mouse coordinate frame

Consider the coordinate frame in the diagram above. When the mouse moves forward, it moves in the positive x direction. When the mouse rotates, it rotates around the z axis (which points upwards). The concept of coordinate frames is fundamental to robotics, but we won't be able to cover it in much depth.[3] Fortunately, we only need to know some basics.

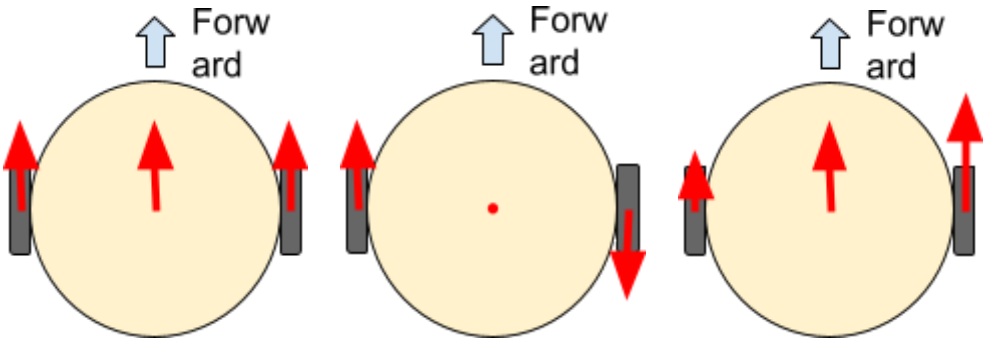
We are mostly interested in the forward velocity and rotational velocity of our mouse. Intuitively, the center of the mouse moves with a velocity that is the average of the wheel velocities. If both wheels move forward at the same speed, the center of the mouse moves forward with the same speed. If the two wheels move in opposite directions at the same speed, the center of the mouse stays in place.[4]

[3] EE106A is a good class to take if you are interested in this kind of stuff.
[4] Try it and see!

We can describe the forward velocity using the following formula:

$$v_f = \frac{1}{2}(v_l + v_r)$$

where v_l is the left wheel velocity and v_r is the right wheel velocity.



Forward velocity

We can also derive a formula for the rotational velocity of the mouse. Just as the forward velocity only depends on the sum (or average) of the two wheel velocities, the rotational velocity only depends on the difference of the two wheel velocities.

The speed of each wheel relative to the center of the mouse is half of the difference between their individual speeds. Rotational velocity is arc velocity divided by radius. We can describe the rotational velocity using the following formula:

$$\omega = \frac{1}{2}(v_r - v_l)/(\frac{1}{2}d) = (v_r - v_l)/d$$

where ω is rotational velocity and d is the distance between the wheels (`WHEELBASE_DIAMETER` in the code).

You will need to translate these formulas into Arduino code.

Arduino Code

```
#define PIN_ENCODER_LEFT_A 11
#define PIN_ENCODER_LEFT_B 2
#define PIN_ENCODER_RIGHT_A 12
#define PIN_ENCODER_RIGHT_B 3

// Mouse physical parameters
const float ENCODER_TICKS_PER_REVOLUTION = 420.0 / 2.0; // blaze it
const float WHEELBASE_DIAMETER = 95.0; // mm
const float WHEEL_DIAMETER = 34.0; // mm
const float VELOCITY_COEFF = WHEEL_DIAMETER * PI / ENCODER_TICKS_PER_REVOLUTION * 1000000.0;
```

```
// Encoder helper variables
unsigned long prev_pulse_time_right;
unsigned long prev_pulse_time_left;

// Encoder state variables
long ticks_left = 0; // ticks
long ticks_right = 0; // ticks
double velocity_left = 0; // millimeters/sec
double velocity_right = 0; // millimeters/sec

double velocity_forward;
double velocity_turn;

int count = 0;

void setup() {
  Serial.begin(9600);

  // Encoder setup
  pinMode(PIN_ENCODER_LEFT_A, INPUT);
  pinMode(PIN_ENCODER_LEFT_B, INPUT);
  pinMode(PIN_ENCODER_RIGHT_A, INPUT);
  pinMode(PIN_ENCODER_RIGHT_B, INPUT);
  attachInterrupt(digitalPinToInterrupt(PIN_ENCODER_LEFT_B), leftEncoderRisingEdge, RISING);
  attachInterrupt(digitalPinToInterrupt(PIN_ENCODER_RIGHT_B), rightEncoderRisingEdge, RISING);
}

void loop() {
  checkEncodersZeroVelocity();

  velocity_forward = /* YOUR CODE HERE */;
  velocity_turn = /* YOUR CODE HERE */;

  if (count % 1000 == 0) {
    // Print debug info every 1000 loops
    Serial.print(velocity_turn);
    Serial.print(" ");
    Serial.println(velocity_forward / 100.0); // scale the forwards velocity so it's easier to
visualize next to the turning velocity
  }
  count++;
}


//////////////////////
// Helper functions //
//////////////////////

void checkEncodersZeroVelocity(void) {
  // Sets the wheel velocity to 0 if we haven't see an edge in a while
  unsigned long curr_time = micros();
  if (curr_time - prev_pulse_time_left > 100000) {
    velocity_left = 0;
  }
  if (curr_time - prev_pulse_time_right > 100000) {
    velocity_right = 0;
  }
}

void leftEncoderRisingEdge(void) {
  unsigned long curr_time = micros();

  int direction;
  if (digitalRead(PIN_ENCODER_LEFT_A) == HIGH) {
    direction = 1;
```

```
  } else {
    direction = -1;
  }

  if (direction * velocity_left < 0) {
    velocity_left = 0;
  } else {
    // Otherwise, convert the period of our pulse in mm/second
    velocity_left = direction * VELOCITY_COEFF / (curr_time - prev_pulse_time_left);
  }
  ticks_left += direction;

  prev_pulse_time_left = curr_time;
}

void rightEncoderRisingEdge(void) {
  unsigned long curr_time = micros();

  int direction;
  if (digitalRead(PIN_ENCODER_RIGHT_A) == HIGH) {
    direction = -1;
  } else {
    direction = 1;
  }

  if (direction * velocity_right < 0) {
    velocity_right = 0;
  } else {
    // Otherwise, convert the period of our pulse in mm/second
    velocity_right = direction * VELOCITY_COEFF / (curr_time - prev_pulse_time_right);
  }
  ticks_right += direction;

  prev_pulse_time_right = curr_time;
}
```

Checkoff #2

1. How many millimeters does a wheel travel per encoder tick?
2. The `checkEncodersZeroVelocity()` function sets wheel velocities to zero if it hasn't seen an encoder pulse in 100 milliseconds. Why do we need this? Will the interrupt callbacks (`leftEncoderRisingEdge` or `rightEncoderRisingEdge`) ever set a wheel velocity to 0?
3. In the code above, use the individual wheel velocities (in mm/sec) to calculate the forward velocity (in mm/sec) and turning velocity (in rad/sec). Visualize these in your **Serial Plotter**.