SeedLab Documentation:

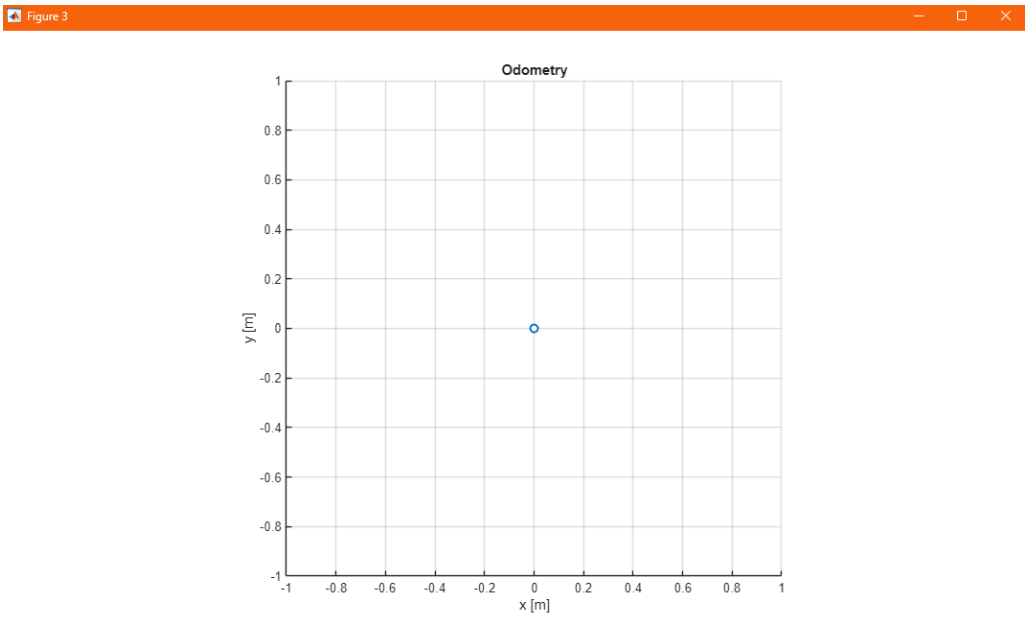
Documentation 2b:

In this assignment the goal was to create an accurate moving model that replicated the robots movement based on its physical components. The code starts by characterizing pins, then setting important measurements into variables that we can use later. These measurements include things like motor encoder counts, wheel size/circumference, and mid-wheel to mid-wheel length. We then fire up our ISR reader to evaluate the counting and movement of the wheels. Based on the movement seen it is printed into a serial monitor (in this case MATLAB) to be live sampled by our MATLAB code creating an animated plot based off the x, y, and angle calculations done in the rest of the Arduino code.

Result is a live graph that plots the x and y as well as the trail and orientation (with a triangle facing starting with the orientation looking +x dir.)

The results are adequately displayed and initial testing confirms correct movement, but further assignments will hopefully improve engine function that will eliminate potential drift and error in the real position.

Results:



Arduino Code:

```
// Demo 2b: Manual odometry only (motors OFF)
// Prints: time_s \t x_m \t y_m \t phi_rad

// ---- Motor driver pins (kept OFF) ----
const int EN         = 4;   // keep LOW == motor off
const int leftMdir   = 7;   // not used
const int leftMpwm   = 9;   // stay 0
const int rightMdir  = 8;   // not used
const int rightMpwm  = 10;  // stay 0

// ---- Encoder pins (UNO) ----
const int leftEncA   = 2;   // interrupt 0
const int leftEncB   = 5;   // GPIO
const int rightEncA  = 3;   // interrupt 1
const int rightEncB  = 6;   // GPIO

// ---- Geometry & scaling ----
const long  COUNTS_PER_REV = 3200;            // 2π rad = 3200 counts
const float PI8            = 3.14159265f;
const float TWO_PI_F       = 2.0f * PI8;

const unsigned long Ts_ms  = 10;              // 100 Hz
const float Ts_s           = Ts_ms / 1000.0f;

const float WHEEL_RADIUS_M = 0.0762f;         // 3 in → meters
const float WHEEL_BASE_M   = 0.36195f;        // 14.25 in → meters

// ---- Per-wheel direction signs ----
// If forward motion still goes the wrong way for a wheel, flip its sign.
const int LEFT_SIGN  = -1;   // -1 first (left reversed)
const int RIGHT_SIGN = +1;   // right looked normal

// ---- State ----
volatile long leftCount  = 0;
volatile long rightCount = 0;

float x_m = 0.0f, y_m = 0.0f, phi = 0.0f;    // phi wrapped to [0, 2π)


// ---- Helpers ----
static inline float wrap_0_2pi(float a){
  while (a < 0.0f)      a += TWO_PI_F;
  while (a >= TWO_PI_F) a -= TWO_PI_F;
```

```
  return a;
}

// ---- Encoder ISRs (standard rule; no baked-in inversion) ----
void ISR_leftA()  {
  int A = digitalRead(leftEncA);
  int B = digitalRead(leftEncB);
  leftCount  += (A == B) ? +1 : -1;
}
void ISR_rightA() {
  int A = digitalRead(rightEncA);
  int B = digitalRead(rightEncB);
  rightCount += (A == B) ? +1 : -1;
}

void setup() {
  // Ensure motors are OFF
  pinMode(EN, OUTPUT);          digitalWrite(EN, LOW);
  pinMode(leftMdir, OUTPUT);    pinMode(leftMpwm, OUTPUT);
  pinMode(rightMdir, OUTPUT);   pinMode(rightMpwm, OUTPUT);
  analogWrite(leftMpwm, 0);     analogWrite(rightMpwm, 0);

  // Encoders
  pinMode(leftEncA, INPUT_PULLUP);
  pinMode(leftEncB, INPUT_PULLUP);
  pinMode(rightEncA, INPUT_PULLUP);
  pinMode(rightEncB, INPUT_PULLUP);

  attachInterrupt(digitalPinToInterrupt(leftEncA),  ISR_leftA,  CHANGE);
  attachInterrupt(digitalPinToInterrupt(rightEncA), ISR_rightA, CHANGE);

  Serial.begin(115200);
  Serial.println("# time_s\tx_m\ty_m\tphi_rad");
}

void loop() {
  static unsigned long t0 = millis(), tLast = t0;
  unsigned long now = millis();
  if (now - tLast < Ts_ms) return;
  tLast = now;
  float time_s = (now - t0) / 1000.0f;

  // --- atomic snapshot of counts ---
  long cL, cR;
  noInterrupts();
```

```
  cL = leftCount;
  cR = rightCount;
  interrupts();

  // --- deltas since last sample ---
  static long pL = 0, pR = 0;
  long dCL = cL - pL;
  long dCR = cR - pR;
  pL = cL;  pR = cR;

  // --- counts -> wheel angle increments (rad), apply per-wheel sign here ---
  const float radPerCount = TWO_PI_F / (float)COUNTS_PER_REV;
  float dThL = (LEFT_SIGN  * dCL) * radPerCount;
  float dThR = (RIGHT_SIGN * dCR) * radPerCount;

  // --- wheel arc lengths (m) ---
  float dSL = WHEEL_RADIUS_M * dThL;
  float dSR = WHEEL_RADIUS_M * dThR;

  // --- differential-drive kinematics ---
  float dS   = 0.5f * (dSL + dSR);           // center distance moved
  float dPhi = (dSR - dSL) / WHEEL_BASE_M;   // heading change

  // midpoint integration for better small-angle accuracy
  float phi_mid = wrap_0_2pi(phi + 0.5f * dPhi);
  x_m  += dS * cosf(phi_mid);
  y_m  += dS * sinf(phi_mid);
  phi   = wrap_0_2pi(phi + dPhi);

  // --- log in requested format ---
  Serial.print(time_s, 3);  Serial.print('\t');
  Serial.print(x_m, 6);     Serial.print('\t');
  Serial.print(y_m, 6);     Serial.print('\t');
  Serial.println(phi, 6);
}
```

Mat Lab Code:

```
% live_stream_plot_universal.m

% Reads Arduino lines:  t  x  y  phi   (tabs or spaces) and animates pose.
```

```
PORT = "COM3";      % <-- your port
BAUD = 115200;

useNew = ~isempty(which('serialport'));   % detect API

% ---- open the port safely ----
if useNew
    s = serialport(PORT, BAUD);
    configureTerminator(s, "LF");
    flush(s);
else
    % close any lingering objects on this port
    old = instrfind('Port', char(PORT));
    if ~isempty(old), try fclose(old); end, try delete(old); end, end
    s = serial(char(PORT), 'BaudRate', BAUD, 'Terminator', 'LF');
    fopen(s);
end
cleaner = onCleanup(@() closeSerialSafe(s, useNew));  % proper cleanup

% ---- plotting setup ----
figure('Color','w'); axis equal; grid on; hold on;
xlabel('x [m]'); ylabel('y [m]'); title('Odometry Live');
xlim([-1 1]); ylim([-1 1]);

trail     = animatedline('LineWidth',1.2);
robotBody = plot(nan, nan, '-', 'LineWidth', 2);
poseDot   = plot(nan, nan, 'o', 'MarkerSize', 6, 'LineWidth', 1.5);

% simple arrow body (about 0.2 m long)
shape = 0.10 * [ 2, -1, -1,  2;
                 0, -0.7, 0.7, 0 ];
transform = @(x,y,th,shp) [ x + (cos(th)*shp(1,:) - sin(th)*shp(2,:));
                            y + (sin(th)*shp(1,:) + cos(th)*shp(2,:)) ];

disp('Reading lines:  t  x  y  phi   (tabs/spaces).  Press Ctrl-C to stop.');

% ---- read loop ----
while true
    % fetch one line (skip comments/empties)
    if useNew
        if s.NumBytesAvailable == 0, pause(0.01); continue; end
        line = strtrim(readline(s));
    else
        if s.BytesAvailable == 0, pause(0.01); continue; end
        line = strtrim(fgetl(s));
    end
    if isempty(line) || line(1) == '#', continue; end

    % parse 4 floats regardless of tab/space separation
    vals = sscanf(line, '%f%f%f%f');
    if numel(vals) ~= 4, continue; end
    t = vals(1); x = vals(2); y = vals(3); phi = vals(4);

    % update trail and robot glyph
    addpoints(trail, x, y);
```

```
    P = transform(x, y, phi, shape);
    set(robotBody, 'XData', P(1,:), 'YData', P(2,:));
    set(poseDot,   'XData', x, 'YData', y);

    % auto-resize axes as the robot moves
    xl = xlim; yl = ylim; pad = 0.2;
    if x<xl(1) || x>xl(2) || y<yl(1) || y>yl(2)
        xlim([min(xl(1),x)-pad, max(xl(2),x)+pad]);
        ylim([min(yl(1),y)-pad, max(yl(2),y)+pad]);
    end

    drawnow limitrate;
end

% ---------- helper for safe cleanup (no anonymous try/catch) ----------
function closeSerialSafe(s, useNew)
    if nargin < 2, useNew = ~isempty(which('serialport')); end
    if useNew
        try flush(s); catch, end
        try delete(s); catch, end
    else
        try
            if strcmp(get(s, 'Status'), 'open'), fclose(s); end
        catch
        end
        try delete(s); catch, end
    end
end
```

Mat Lab Results: