2D Bang-Bang Control (Finding Minimum Ray)

ROB 102: Introduction to AI & Programming

2021/09/22

Administrative

Project 0 & Project 1 due **October 4th, at 11:59 PM**.

Deliverables: Project 0

- Code on GitHub (tag the final version)
- In-class Demo (Oct 6th)

Deliverables: Project 1

- Code on GitHub (tag the final version)
- In-class Demo (Oct 6th)
- Demo video of all parts (linked in README)

Administrative

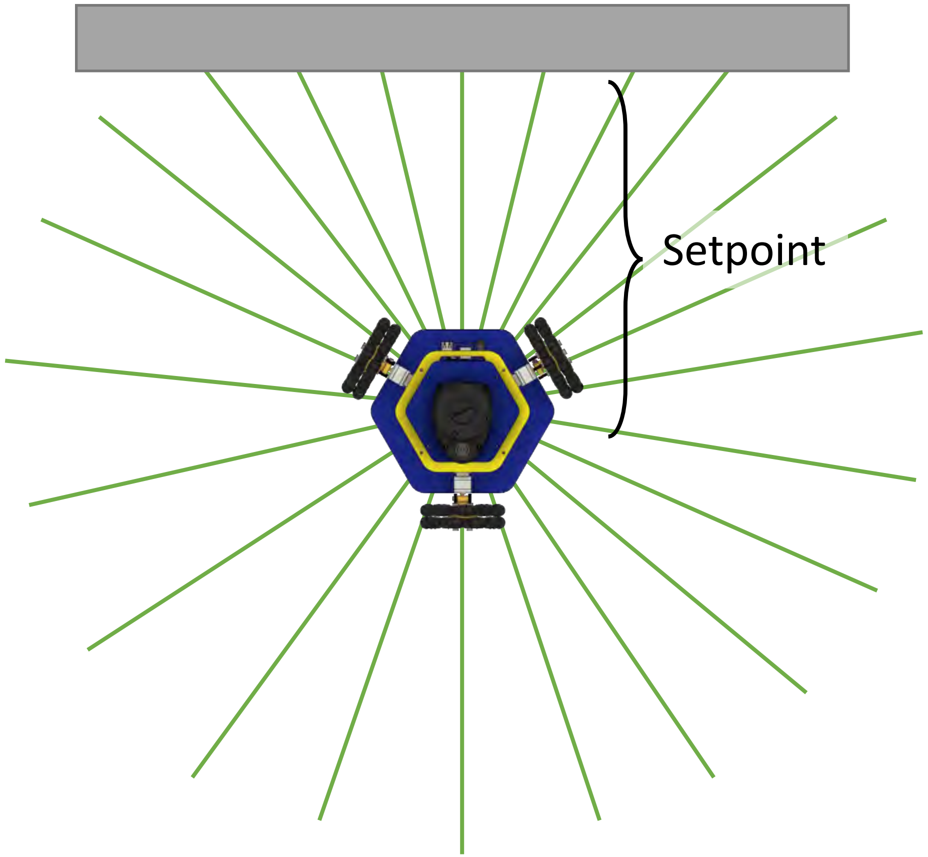
Project 0 & Project 1 due **October 4th, at 11:59 PM**.

Demo Day for Project 0 & Project 1: **October 6th, in class** (FRB 2000)

- All team members should be present
- Send instructors a message if you can't attend
- Your robot should complete a full lap around the demo course
- P0 will be demo-ed individually on student laptop

Friday's lab: Driving parallel to the wall (for Project 1)

Last Week: 1D Control Problem

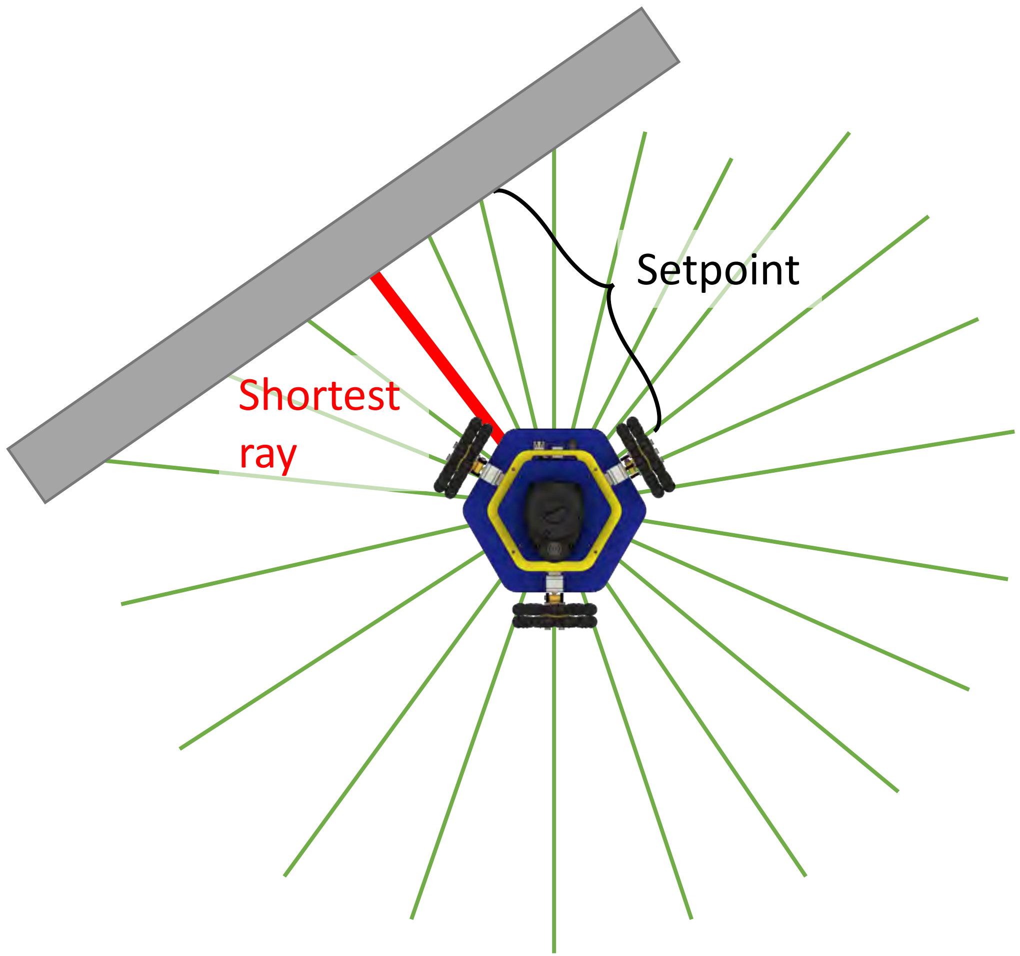


Goal: Write a controller so that the robot drives towards the wall and stops a certain distance from the wall.

The desired distance from the wall is called the **setpoint**.

Maintains distance to the wall directly in front of the robot.

Today: Bang-Bang Control to Nearest Wall

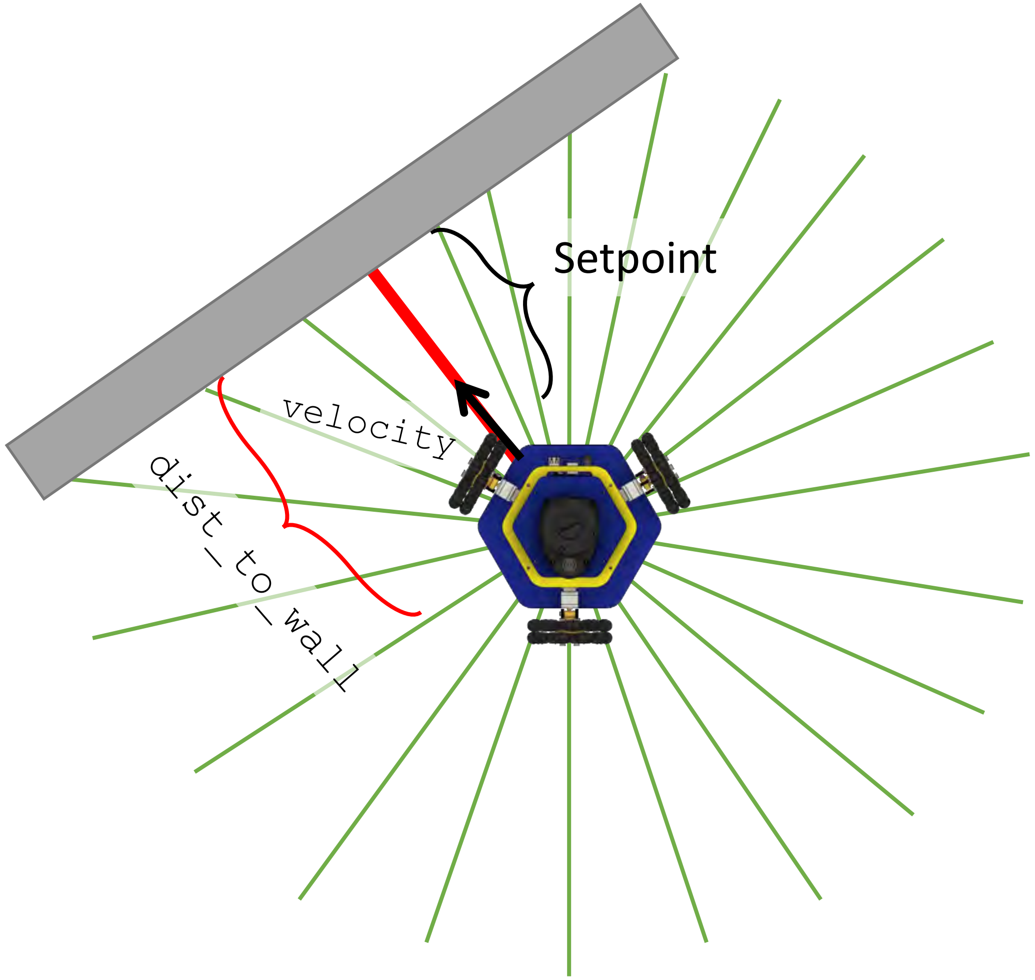


Goal: Write a controller so that the robot maintains a distance to the *nearest* wall.

How? We can follow the *shortest ray*.

In Project 1, we will follow the wall by driving *perpendicular* to the shortest ray.

Today: Bang-Bang Control to Nearest Wall



We can use the same controller (bang-bang or P-control) as last week. But this time, we'll drive in the direction of the shortest ray.

We need to:

1. Find the direction and length of the shortest ray,
2. Drive the robot in any direction.

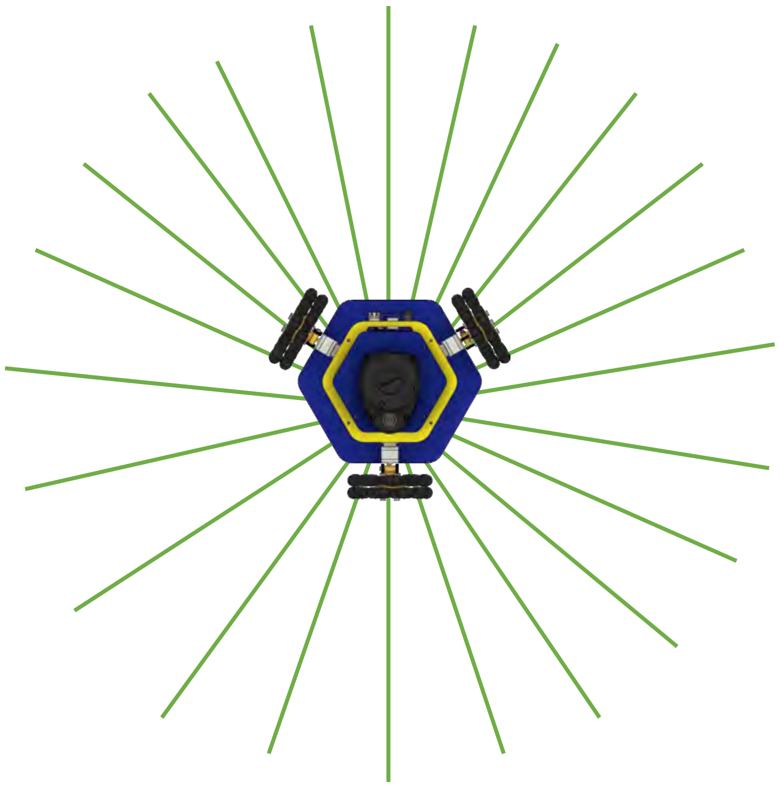
Recall: Laser scan data

The Lidar sends out a series of rays. The LidarScan data type is a struct of vectors.

```
struct LidarScan
{
    bool good;

    int utime;
    int num_ranges;

    std::vector<float> ranges;
    std::vector<float> thetas;
    std::vector<float> intensities;
    std::vector<float> times;
};
```

- `bool good;` ← Whether scan is valid
- `int num_ranges;` ← Number of rays in the scan
- `ranges` ← Ray ranges (lengths, in meters)
- `thetas` ← Ray angles (in radians)
- `intensities` ← Ray intensities
- `times` ← Ray times

These vectors have length `num_ranges`



Finding the minimum length ray

main.cpp ×

```cpp
#include <iostream>
#include <vector>

int main() {
    std::vector<int> v = {71, 9, 80, 15, 6, 52};

    int min_idx = 0;
    for (int i = 0; i < v.size(); i++)
    {
        Your code here
    }

    std::cout << "Min index: " << min_idx << "\n";
    std::cout << "Min value: " << v[min_idx] << "\n";
}
```

Minimum value

Retrieving the value at the index gives the minimum value

Goal: Find the *index* of the minimum length ray.

Console Shell

```
~/Test$ g++ main.cpp -std=c++11 -o main
~/Test$ ./main
Min index: 4
Min value: 6
~/Test$
```

Minimum value is at index 4

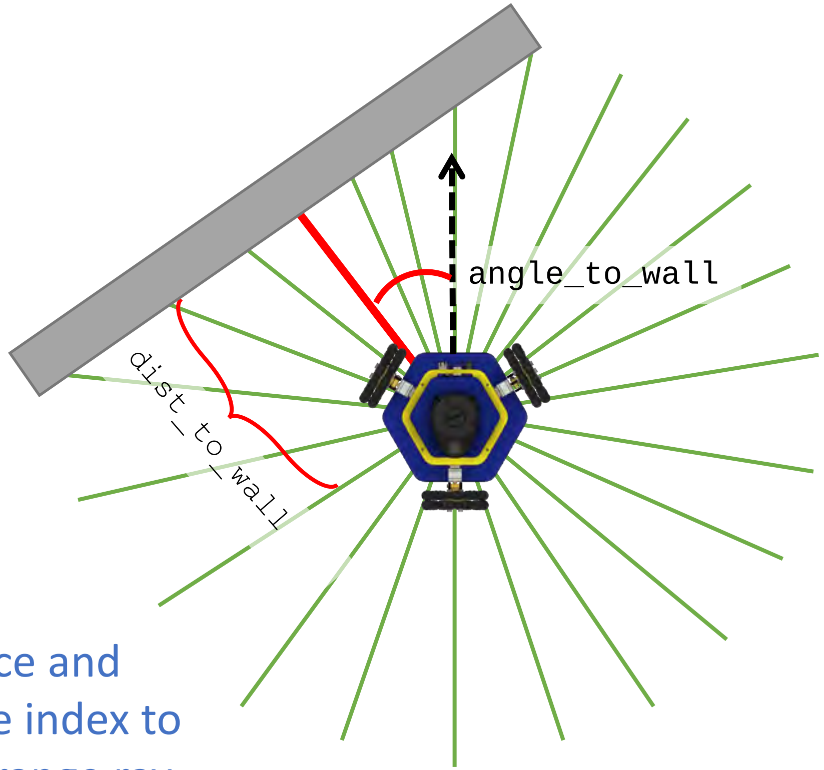
Finding the minimum length ray

```
struct LidarScan
{
    bool good;

    int utime;
    int num_ranges;

    std::vector<float> ranges;
    std::vector<float> thetas;
    std::vector<float> intensities;
    std::vector<float> times;
};
```

The minimum length ray is the one with the smallest range value

```
// Get the distance to the wall.
float min_idx = findMinDist(scan);
float dist_to_wall = scan.ranges[min_idx];
float angle_to_wall = scan.thetas[min_idx];
```

Goal: Find the index of the minimum length ray.



Get the distance and angle using the index to the minimum range ray

Today: Min length ray

```
float dt = 0.01;          // seconds
float setpoint = 0.35;  // meters

while (true) {
    LidarScan scan = readLidarScan(drv);

    if (scan.good)
    {
        // Get the distance to the wall.
        float min_idx = findMinDist(scan);
        float dist_to_wall = scan.ranges[min_idx];
        float angle_to_wall = scan.thetas[min_idx];

        std::cout << "Min distance: " << dist_to_wall << " Angle: " << angle_to_wall;
        std::cout << " Intensity: " << scan.intensities[min_idx] << " | ";

        // Calculate the appropriate control signal.
        float vel = feedbackControl(dist_to_wall, setpoint);

        std::cout << "Setpoint: " << setpoint << " Velocity: " << vel;

        // Apply the control signal.
        float vx = 0;
        float vy = 0;
        /**
         * TODO: Use the angle to the wall (angle_to_wall) to decompose the
         * velocity command (vel) into its x and y components. Store these
         * in vx and vy respectively.
         **/
        std::cout << " (vx, vy): (" << vx << ", " << vy << ")\n";

        drive(vx, vy, 0);
    }

    sleepFor(dt);

    if (ctrl_c_pressed) break;
}
```

Setpoint in meters

Loop forever

Read a scan

Find index of minimum distance
(Your code!)

Grab distance and angle to wall

Get control signal
(Your code! From last week)

Decompose velocity into x and y components
(Your code!)

Send the velocity signal to the robot

Today: Min length ray

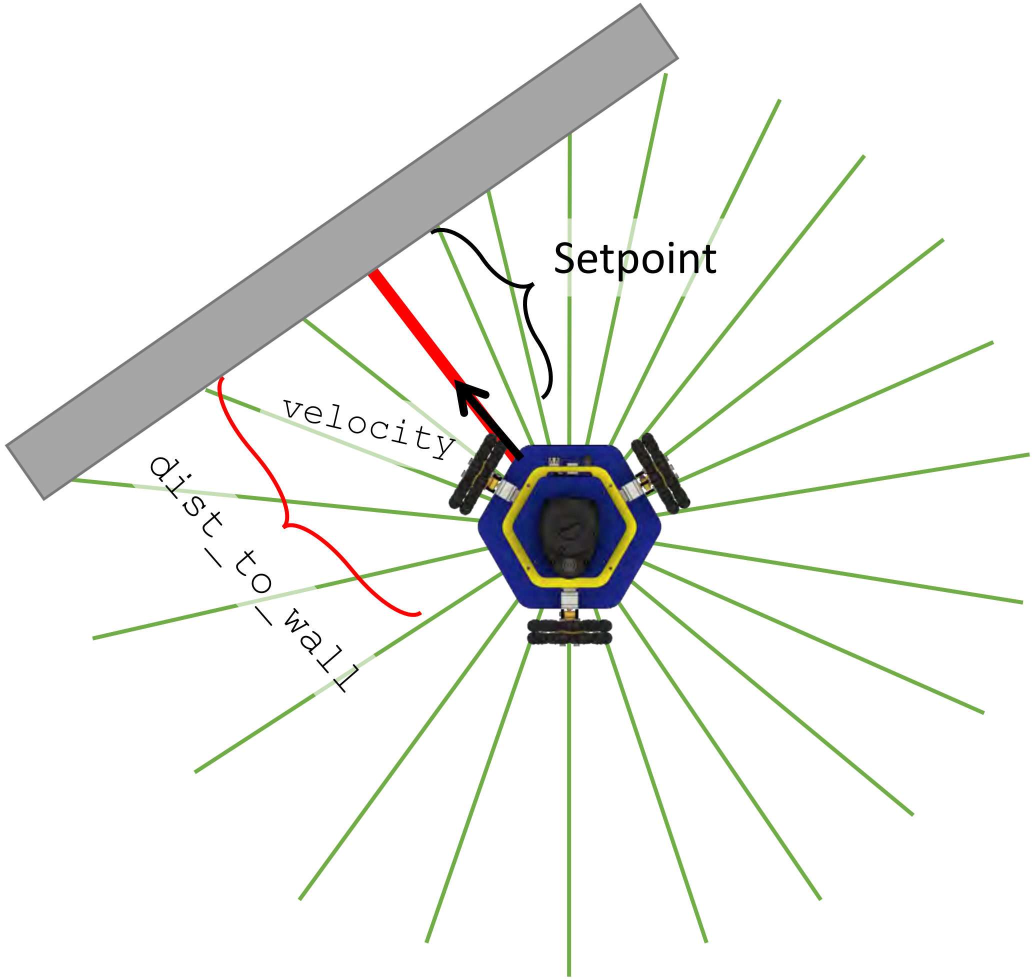
```cpp
float dt = 0.01;         // seconds
float setpoint = 0.35;  // meters

while (true) {
    LidarScan scan = readLidarScan(drv);

    if (scan.good)
    {
        // Get the distance to the wall.
        float min_idx = findMinDist(scan);
        float dist_to_wall = scan.ranges[min_idx];
        float angle_to_wall = scan.thetas[min_idx];

        std::cout << "Min distance: " << dist_to_wall << " Angle: " << angle_to_wall;
        std::cout << " Intensity: " << scan.intensities[min_idx] << " | ";

        // Calculate the appropriate control signal.
        float vel = feedbackControl(dist_to_wall, setpoint);

        std::cout << "Setpoint: " << setpoint << " Velocity: " << vel;

        // Apply the control signal.
        float vx = 0;
        float vy = 0;
        /**
         * TODO: Use the angle to the wall (angle_to_wall) to decompose the
         * velocity command (vel) into its x and y components. Store these
         * in vx and vy respectively.
         **/
        std::cout << " (vx, vy): (" << vx << ", " << vy << ")\n";

        drive(vx, vy, 0);
    }

    sleepFor(dt);

    if (ctrl_c_pressed) break;
}
```

```cpp
int findMinDist(const LidarScan& scan)
{
    int min_idx = 0;
    /**
     * TODO: Return the index of the shortest ray in the Lidar scan. For
     * example, if the shortest ray is the third one, at index 2, return 2.
     *
     * HINT: The length of each ray is stored in the vector scan.ranges.
     *
     * HINT: Do not take into account any rays which have 0 intensity. Those rays
     * will have default range 0, which will always be the minimum if you forget
     * to check the intensity. The intensities are stored in scan.intensities.
     **/
    return min_idx;
}
```

TODO (1): Write a function to find the index of the minimum length ray in a given Lidar scan.

Make sure to ignore any rays with zero intensity (those default to zero range)

Today: Bang-Bang Control to Nearest Wall



We can use the same controller (bang-bang or P-control) as last week. But this time, we'll drive in the direction of the shortest ray.

We need to:

1. Find the direction and length of the shortest ray,
2. Drive the robot in any direction.

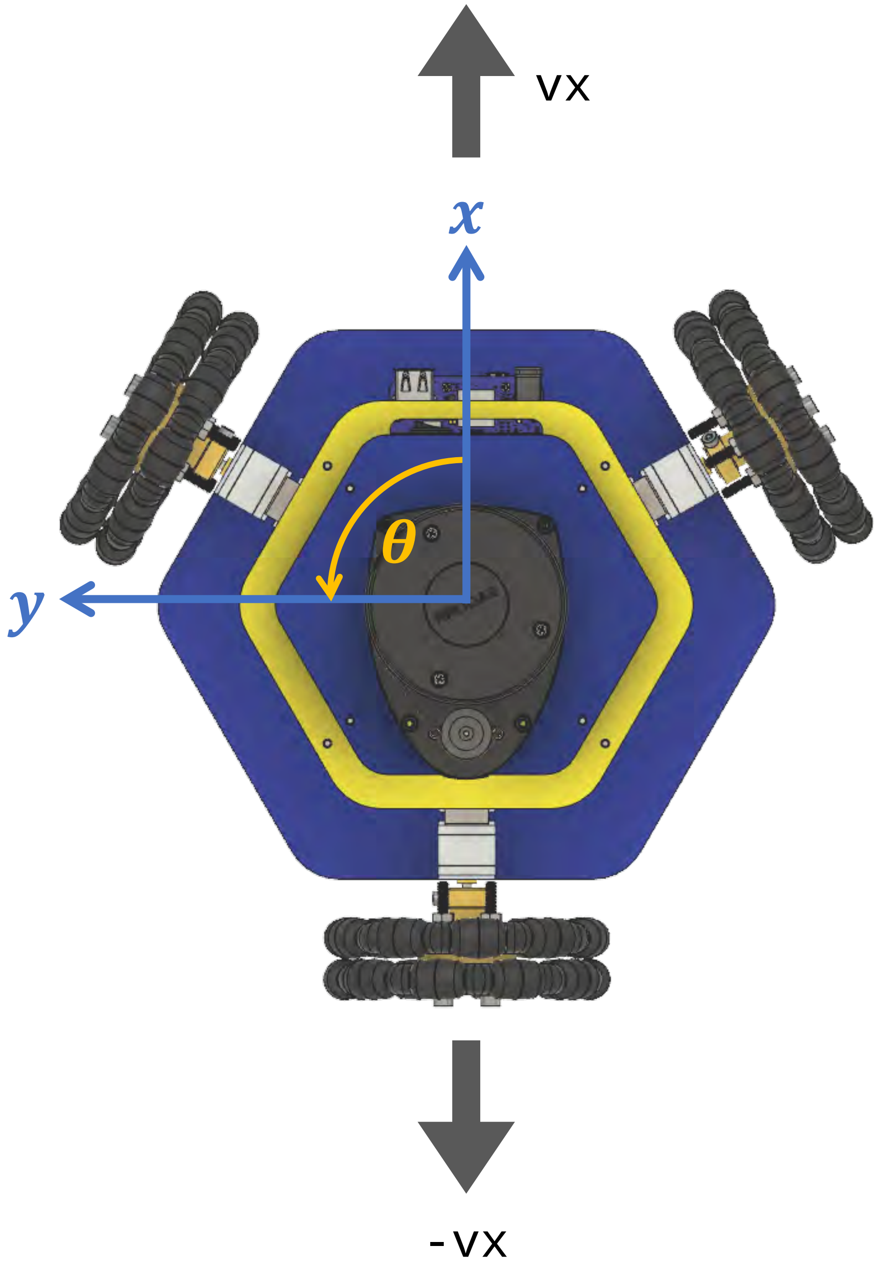
2D Velocity Control

Moving the robot forward:

```
drive(vx, 0, 0);
```

Moving the robot backward:

```
drive(-vx, 0, 0);
```



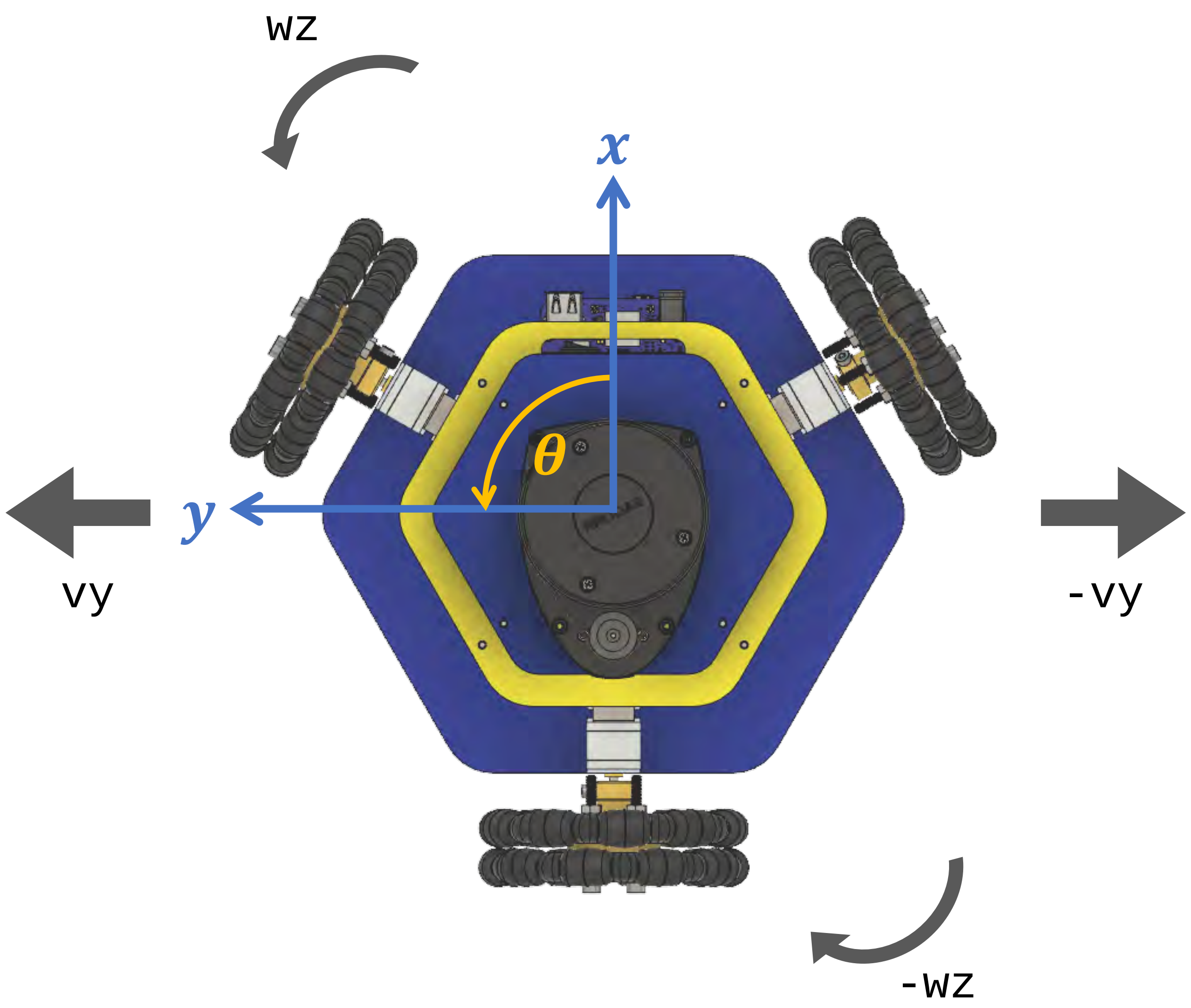
2D Velocity Control

Moving the robot left:

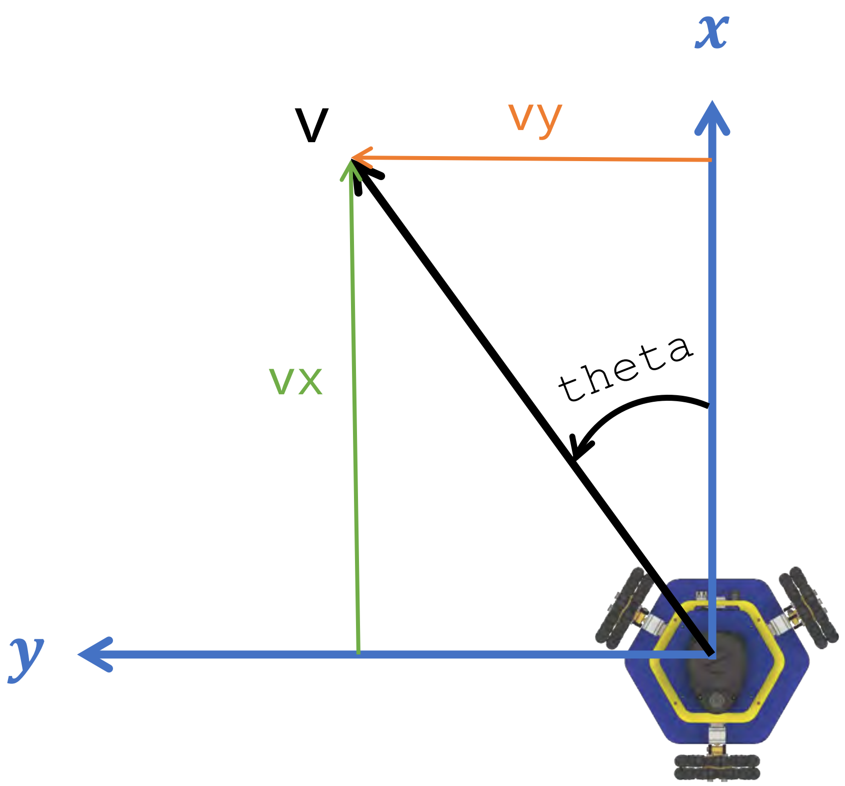
```
drive(0, vy, 0);
```

Rotating counterclockwise:

```
drive(0, 0, wz);
```



2D Velocity Control: Trigonometry Review



To move in any direction (no rotation):

```
vx = v * cos(theta)
vy = v * sin(theta)

drive(vx, vy, 0)
```

This will work for any velocity and angle (try it yourself!)

The <cmath> library

main.cpp

```cpp
#include <iostream>
#include <cmath>

int main() {
  // Common math expressions
  std::cout << "pow(3, 3) = " << pow(3, 3) << "\n";
  std::cout << "sqrt(2) = " << sqrt(2) << "\n";
  std::cout << "abs(-3) = " << abs(-3) << "\n";
  std::cout << "fabs(-1.5) = " << fabs(-1.5) << "\n";
  std::cout << "Careful! abs(-1.5) = " << abs(-1.5) << "\n";

  // Trig functions
  float pi = 3.14159265359;
  std::cout << "\nsin(1.0) = " << sin(1.0) << "\n";
  std::cout << "tan(pi) = " << tan(pi) << "\n";

  // Logs & Exponentials
  float e = 2.71828;
  std::cout << "\nlog(e) = " << log(e) << "\n";
  std::cout << "exp(1.0) = " << exp(1.0) << "\n";
}
```

Remember to include the library!

Contains common math operations.

Console Shell

```
~/Test$ g++ main.cpp -o main
~/Test$ ./main
pow(3, 3) = 27
sqrt(2) = 1.41421
abs(-3) = 3
fabs(-1.5) = 1.5
Careful! abs(-1.5) = 1

sin(1.0) = 0.841471
tan(pi) = 8.74228e-08

log(e) = 0.999999
exp(1.0) = 2.71828
~/Test$
```

3^3

$\sqrt{2}$

$|-1.5|$

$e^{1.0}$

The <cmath> library

For a list of all the functions:

https://www.cplusplus.com/reference/cmath/

This website is a great reference for all things C++!

fx Functions

Trigonometric functions

cos	Compute cosine (function)
sin	Compute sine (function)
tan	Compute tangent (function)
acos	Compute arc cosine (function)
asin	Compute arc sine (function)
atan	Compute arc tangent (function)
atan2	Compute arc tangent with two parameters (function)

Hyperbolic functions

cosh	Compute hyperbolic cosine (function)
sinh	Compute hyperbolic sine (function)
tanh	Compute hyperbolic tangent (function)
acosh C++11	Compute area hyperbolic cosine (function)
asinh C++11	Compute area hyperbolic sine (function)
atanh C++11	Compute area hyperbolic tangent (function)

Exponential and logarithmic functions

exp	Compute exponential function (function)
frexp	Get significand and exponent (function)
ldexp	Generate value from significand and exponent (function)
log	Compute natural logarithm (function)
log10	Compute common logarithm (function)
modf	Break into fractional and integral parts (function)
exp2 C++11	Compute binary exponential function (function)
expm1 C++11	Compute exponential minus one (function)
ilogb C++11	Integer binary logarithm (function)
log1p C++11	Compute logarithm plus one (function)
log2 C++11	Compute binary logarithm (function)
logb C++11	Compute floating-point base logarithm (function)
scalbn C++11	Scale significand using floating-point base exponent (function)
scalbln C++11	Scale significand using floating-point base exponent (long) (function)

Today: 2D Control Code

```
float dt = 0.01;        // seconds
float setpoint = 0.35;  // meters

while (true) {
    LidarScan scan = readLidarScan(drv);

    if (scan.good)
    {
        // Get the distance to the wall.
        float min_idx = findMinDist(scan);
        float dist_to_wall = scan.ranges[min_idx];
        float angle_to_wall = scan.thetas[min_idx];

        std::cout << "Min distance: " << dist_to_wall << " Angle: " << angle_to_wall;
        std::cout << " Intensity: " << scan.intensities[min_idx] << " | ";

        // Calculate the appropriate control signal.
        float vel = feedbackControl(dist_to_wall, setpoint);

        std::cout << "Setpoint: " << setpoint << " Velocity: " << vel;

        // Apply the control signal.
        float vx = 0;
        float vy = 0;
        /**
         * TODO: Use the angle to the wall (angle_to_wall) to decompose the
         * velocity command (vel) into its x and y components. Store these
         * in vx and vy respectively.
         **/
        std::cout << " (vx, vy): (" << vx << ", " << vy << ")\n";

        drive(vx, vy, 0);
    }

    sleepFor(dt);

    if (ctrl_c_pressed) break;
}
```

Setpoint in meters

Loop forever

Read a scan

Find index of minimum distance
(Your code!)

Grab distance and angle to wall

Get control signal
(Your code! From last week)

Decompose velocity into x and y components
(Your code!)

Send the velocity signal to the robot

Today: Feedback Control

```
    float dt = 0.01;         // seconds
    float setpoint = 0.35;  // meters

    while (true) {
        LidarScan scan = readLidarScan(drv);

        if (scan.good)
        {
            // Get the distance to the wall.
            float min_idx = findMinDist(scan);
            float dist_to_wall = scan.ranges[min_idx];
            float angle_to_wall = scan.thetas[min_idx];

            std::cout << "Min distance: " << dist_to_wall << " Angle: " << angle_to_wall;
            std::cout << " Intensity: " << scan.intensities[min_idx] << " | ";

            // Calculate the appropriate control signal.
            float vel = feedbackControl(dist_to_wall, setpoint);

            std::cout << "Setpoint: " << setpoint << " Velocity: " << vel;

            // Apply the control signal.
            float vx = 0;
            float vy = 0;
            /**
             * TODO: Use the angle to the wall (angle_to_wall) to decompose the
             * velocity command (vel) into its x and y components. Store these
             * in vx and vy respectively.
             **/
            std::cout << " (vx, vy): (" << vx << ", " << vy << ")\n";

            drive(vx, vy, 0);
        }

        sleepFor(dt);

        if (ctrl_c_pressed) break;
    }
```

```
float feedbackControl(float dist_to_wall, float setpoint)
{
    float vel = 0;
    /**
     * TODO: Calculate the control command to send to the robot given the
     * current distance to the wall and the desired setpoint.
     *
     * You can use either Bang-Bang control or P-control. Reuse your code from
     * the 1D control activity.
     **/
    return vel;
}
```

TODO (2): Write a function that returns a control command given the current distance and the setpoint (use bang-bang or P-control).

Reuse your code from last time!

Today: 2D Velocity Commands

```
float dt = 0.01;        // seconds
float setpoint = 0.35;  // meters

while (true) {
    LidarScan scan = readLidarScan(drv);

    if (scan.good)
    {
        // Get the distance to the wall.
        float min_idx = findMinDist(scan);
        float dist_to_wall = scan.ranges[min_idx];
        float angle_to_wall = scan.thetas[min_idx];

        std::cout << "Min distance: " << dist_to_wall << " Angle: " << angle_to_wall;
        std::cout << " Intensity: " << scan.intensities[min_idx] << " | ";

        // Calculate the appropriate control signal.
        float vel = feedbackControl(dist_to_wall, setpoint);

        std::cout << "Setpoint: " << setpoint << " Velocity: " << vel;

        // Apply the control signal.
        float vx = 0;
        float vy = 0;
        /**
         * TODO: Use the angle to the wall (angle_to_wall) to decompose the
         * velocity command (vel) into its x and y components. Store these
         * in vx and vy respectively.
         **/
        std::cout << " (vx, vy): (" << vx << ", " << vy << ")\n";

        drive(vx, vy, 0);
    }

    sleepFor(dt);

    if (ctrl_c_pressed) break;
}
```

TODO (3): Convert the velocity magnitude and the angle into (vx, vy) commands.


```
// Apply the control signal.
float vx = 0;
float vy = 0;
/**
 * TODO: Use the angle to the wall (angle_to_wall) to decompose the
 * velocity command (vel) into its x and y components. Store these
 * in vx and vy respectively.
 **/
std::cout << " (vx, vy): (" << vx << ", " << vy << ")\n";

drive(vx, vy, 0);
```

Today:

1. Accept the assignment for this activity: https://classroom.github.com/g/RkYnescx
2. Clone the repository on your robot
3. Write a function that finds the index of the minimum ray in the Lidar scan in `findMinDist()`
4. Rewrite your control function from last time in `feedbackControl()`
5. Decompose the velocity command into a 2D command
6. Test your code on the robot!

```
int findMinDist(const LidarScan& scan)
{
    Your code here!
}
```

```
float feedbackControl(float dist_to_wall, float setpoint)
{
    Your code here!
}
```

```
// Apply the control signal.
float vx = 0;
float vy = 0;
Your code here!
drive(vx, vy, 0);
```