Intermediate Programming Concepts for FTC

Kanthan Rajendran
Infinity Tech Mentor

INFINITY TECH
FTC TEAM 13684

FTC Canton Kickoff Event

September 7, 2024

FIRST TECH CHALLENGE

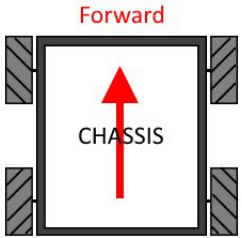
Agenda

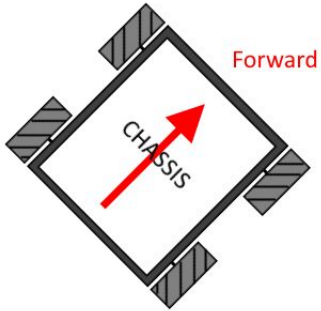
- Field-Centric Mecanum Drive
- PID Control for Motors
- Odometry
- Finite State Machine

Field-Centric Mecanum Drive

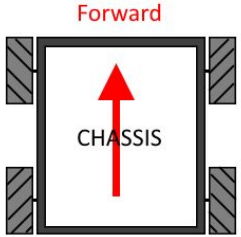
INFINITY TECH
FTC TEAM 13684

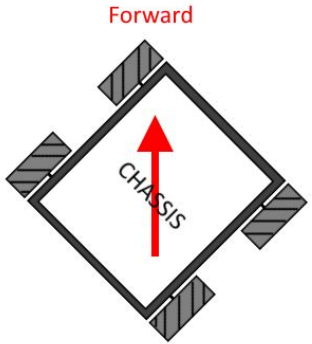
Robot-Centric vs Field-Centric





Robot-Centric
Robot uses orientation relative to itself

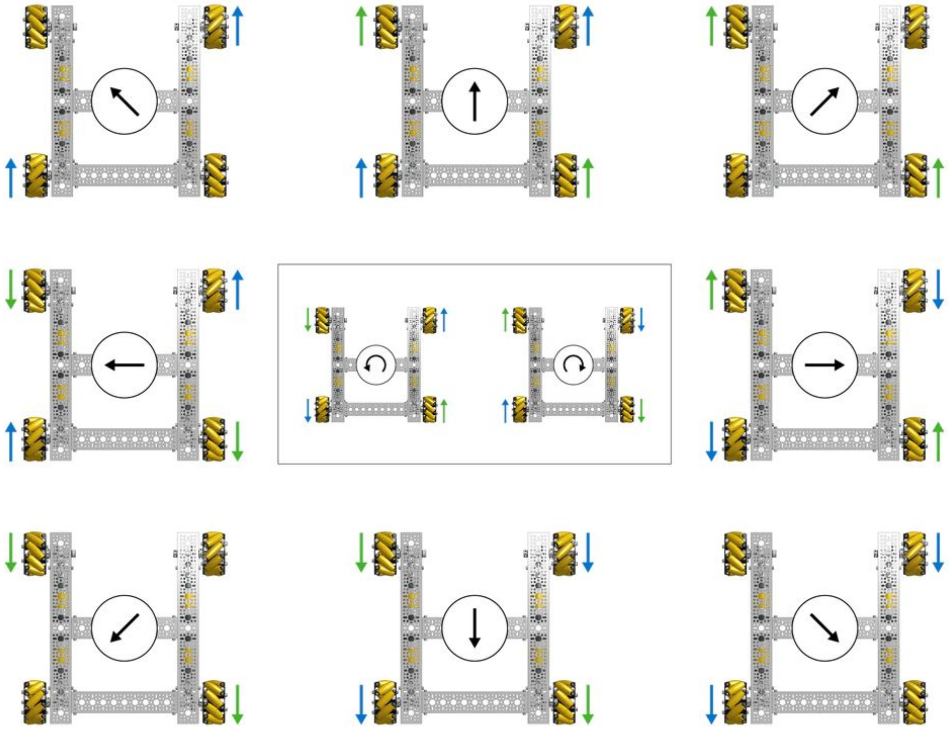




Field-Centric
Robot uses absolute orientation;
Movement directions are relative to field

INFINITY TECH
FTC TEAM 13684

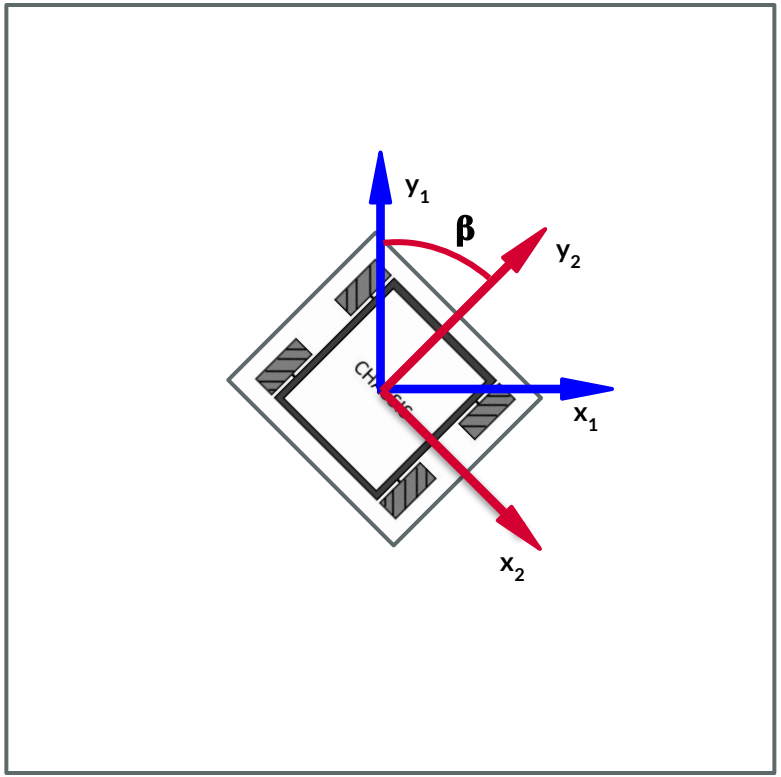
Robot-Centric Mecanum Drive Code



```
double y = -gamepad1.left_stick_y; // Y stick is reversed!
double x = gamepad1.left_stick_x;
double rx = gamepad1.right_stick_x;

frontLeftMotor.setPower(y + x + rx);
backLeftMotor.setPower(y - x + rx);
frontRightMotor.setPower(y - x - rx);
backRightMotor.setPower(y + x - rx);
```

INFINITY TECH
FTC TEAM 13684

2D Coordinate Translation



y_1, x_1 are **field-centric** directions

y_2, x_2 are **robot-centric** directions

β is robot **heading** relative to the field

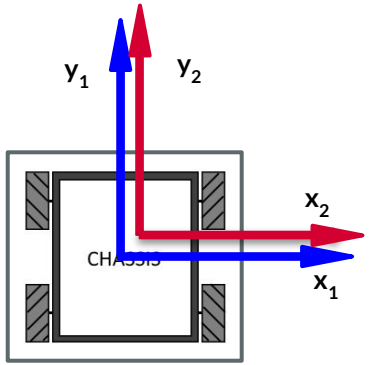
To implement field-centric robot drive, joystick inputs need to be translated into robot-centric coordinate system.

$$x_2 = x_1 cos\beta - y_1 sin\beta$$
$$y_2 = x_1 sin\beta + y_1 cos\beta$$

INFINITY TECH
FTC TEAM 13684

2D Coordinate Translation

$$x_2 = x_1 cos\beta - y_1 sin\beta$$
$$y_2 = x_1 sin\beta + y_1 cos\beta$$

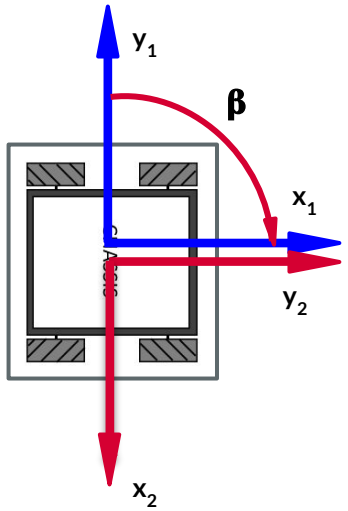


If $\beta = 0°$;

$x_2 = x_1$

$y_2 = y_1$

No translation!



If $\beta = 90°$;

$x_2 = -y_1$

$y_2 = x_1$

If you push joystick forward, robot will actually **strafe to the left**, but **move forward** relative to field

INFINITY TECH
FTC TEAM 13684

Field-Centric Mecanum Drive Code

```
double y = -gamepad1.left_stick_y; // Y stick is reversed!
double x = gamepad1.left_stick_x;
double rx = gamepad1.right_stick_x;

// Rotate the movement direction counter to the bot's rotation
double rotX = x * Math.cos(-botHeading) - y * Math.sin(-botHeading);
double rotY = x * Math.sin(-botHeading) + y * Math.cos(-botHeading);


double frontLeftPower = (rotY + rotX + rx);
double backLeftPower = (rotY - rotX + rx);
double frontRightPower = (rotY - rotX - rx);
double backRightPower = (rotY + rotX - rx);
```

Obtaining Robot Heading

```
// Retrieve the IMU from the hardware map
IMU imu = hardwareMap.get(IMU.class, "imu");
// Adjust the orientation parameters to match your robot
IMU.Parameters parameters = new IMU.Parameters(new RevHubOrientationOnRobot(
        RevHubOrientationOnRobot.LogoFacingDirection.UP,
        RevHubOrientationOnRobot.UsbFacingDirection.FORWARD));
// Without this, the REV Hub's orientation is assumed to be logo up / USB forward
imu.initialize(parameters);

double botHeading = imu.getRobotYawPitchRollAngles().getYaw(AngleUnit.RADIANS);
```

Note: *IMU interface has been updated for latest SDK due to new IMU chip; older code will need to be updated.*

REV Control Hub or **Expansion Hub** (older units before Dec 2021) have built-in **Inertial Measurement Unit (IMU)** that can be used to obtain robot orientation.

Yaw angle measures robot **heading** (side-to-side lateral rotation)

Heading is **initialized** when robot is powered-up. Can be reinitialized to correct for **drift**.

PID Control for Motors

INFINITY TECH
FTC TEAM 13684

FTC Motors

REV HD Hex Motor

REV Core Hex Motor

goBILDA Yellow Jacket Motors

Note: Every FTC legal motor has built-in **encoder**.

INFINITY TECH
FTC TEAM 13684

Running the Motor without Encoders

- Most straightforward approach to run a motor is to set power directly
- Power can be set between -1.0 and 1.0
  - **positive** power value makes motor spin **counterclockwise** when looking towards motor
  - motor direction can be **reversed**

```
motor.setPower(-0.6);
```

- You can also use input from **joystick** to set the power to the motors
- Joystick input also varies between -1.0 and 1.0

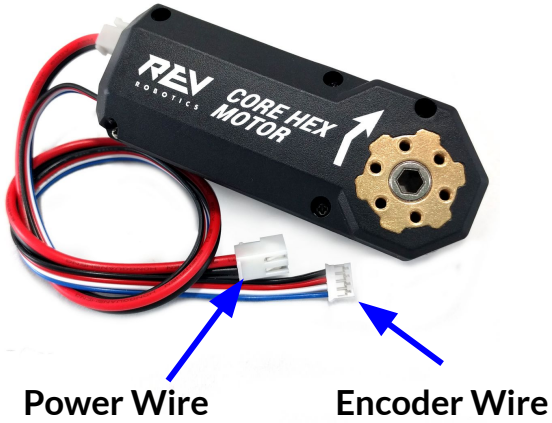
```
double left_power = -gamepad1.left_stick_y;
double right_power = -gamepad1.right_stick_y;

left_drive.setPower(left_power);
right_drive.setPower(right_power);
```

INFINITY TECH
FTC TEAM 13684

What are Encoders?

- **Encoders** are sensors that track rotational movement around an axis
- Used to find the position of the robot or one of its mechanisms
- Separate wire for encoder
- Different motors have a different **CPR** (Counts Per Revolution)
    - Core Hex Motor has 288 CPR (1 count per 1.25°)

```
motor.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);

int position = motor.getCurrentPosition();
```



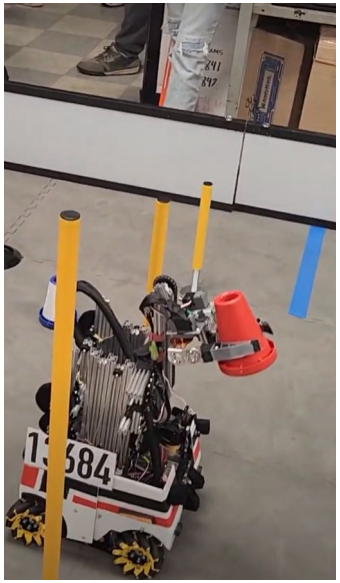
INFINITY TECH
FTC TEAM 13684

Running the Motor using Encoder

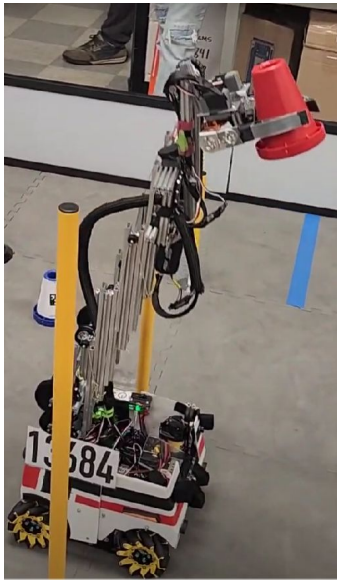
Run To Position

- We can specify the **target encoder value** that we want the motor to run to
- Use the **RUN_TO_POSITION** mode
- This mode is a good choice for mechanisms that require a specific position and can use buttons on a gamepad for control.

```
motor.setMode(DcMotor.RunMode.RUN_TO_POSITION);

motor.setTargetPosition(target_encoder_value);
motor.setPower(power);
```



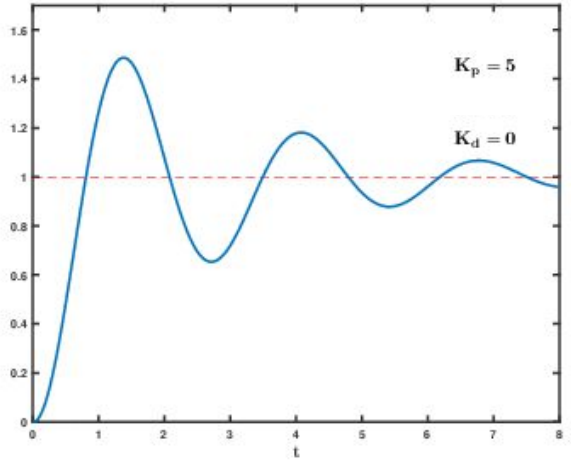
Encoder Position = 0

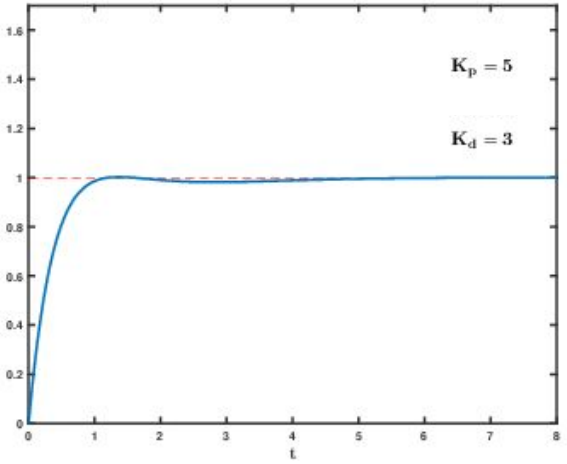


Encoder Position = 400

INFINITY TECH
FTC TEAM 13684

Oscillations





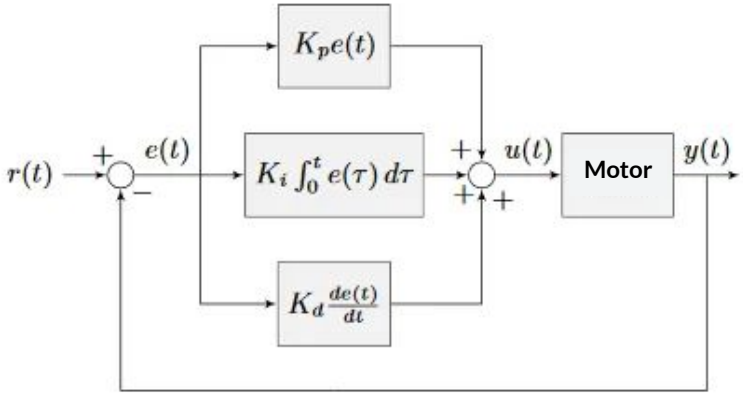
- Without a proper control loop, Run to Position can result in overshoot and undershoot oscillations
- Feedback control loop needed to achieve faster and more stable response.

PID Controller

- **Proportional Integral Derivative** controller
- A form of a **feedback control loop**, or **closed loop control**
- Uses **error** (the difference between target and current positions) to continuously adjust motor power
- Coefficients for each term, K_p, K_i, K_d

$$u(t) = K_\mathrm{p} e(t) + K_\mathrm{i} \int_0^t e(\tau)\,\mathrm{d}\tau + K_\mathrm{d} \frac{\mathrm{d}e(t)}{\mathrm{d}t},$$

r(t) is the target encoder position
y(t) is the current encoder position
e(t) is the error = *r(t)* - *y(t)*
u(t) is the motor power



INFINITY TECH
FTC TEAM 13684

PID Controller

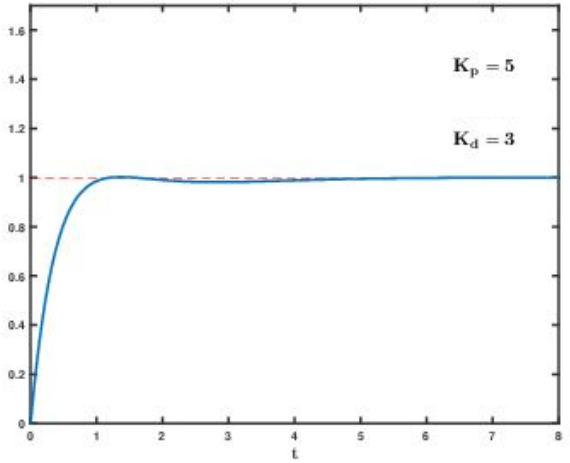
- For FTC applications, integral term generally ignored ($K_i = 0$)
- **Proportional Derivative** controller
- **Proportional** term produces an output value that is proportional to the current error value
  - when error is large, output is large
  - as error decreases, motor power decreases
- **Derivative** term is intended to dampen the rate of change of the error
  - as error decreases, derivative of error becomes negative

$$u(t) = K_p e(t) + K_d \frac{de}{dt}$$

INFINITY TECH
FTC TEAM 13684

PID Pseudocode

```
while True:
    current_time = get_current_time()

    current_error = target_position - current_position

    p = k_p * current_error

    d = k_d * (current_error - previous_error) / (current_time - previous_time)

    output = p + d

    previous_error = current_error
    previous_time = current_time
```



INFINITY TECH
FTC TEAM 13684

Implementation

- Two methods to implement PID
  - Manually program feedback loop
  - Utilize **built-in** PID controller
- PID coefficients can be modified

```
motor.setMode(DcMotor.RunMode.RUN_TO_POSITION);

motor.setMode(DcMotor.RunMode.RUN_USING_ENCODER);

motor.setTargetPosition(target_encoder_value);

motor.setVelocity(max_velocity);
```

```
// Change coefficients using methods included with DcMotorEx class.
PIDFCoefficients pidfNew = new PIDFCoefficients(NEW_P, NEW_I, NEW_D, NEW_F);
motor.setPIDFCoefficients(DcMotor.RunMode.RUN_USING_ENCODER, pidfNew);
```

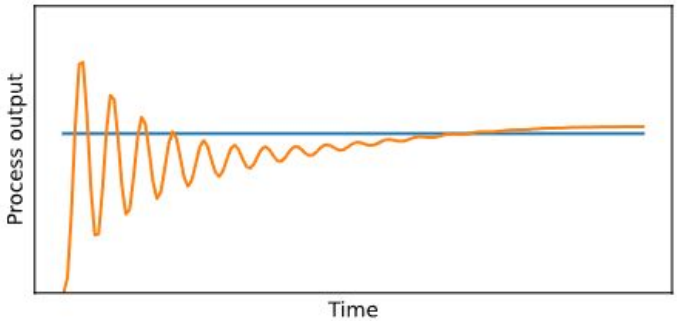
INFINITY TECH
FTC TEAM 13684

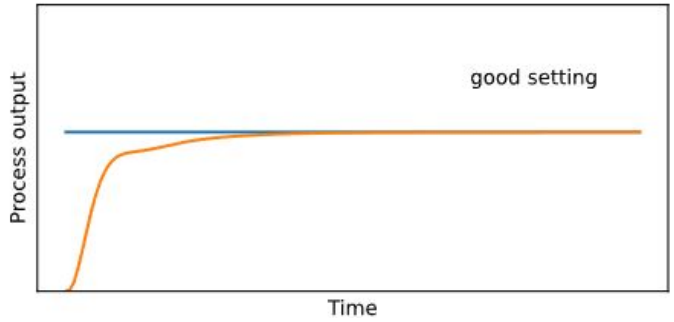
Tuning

Tuning of PID coefficients is critical to achieve good behavior.

The most common method for tuning a PD controller is as follows:

1. Set the D coefficient to zero
2. Increase the P coefficient until there are oscillations around the target
3. Increase the D coefficient until no overshoot occurs





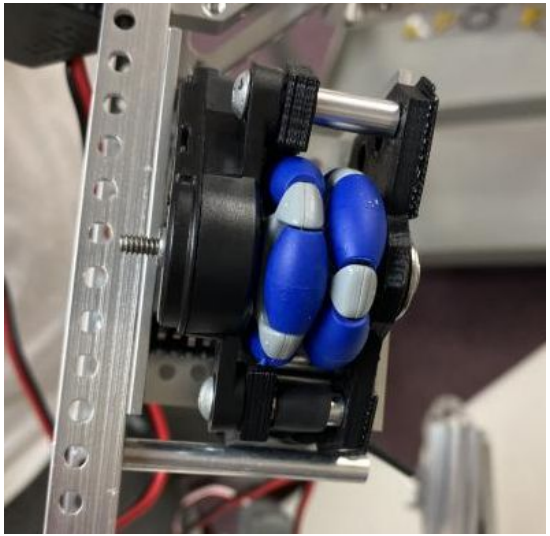
INFINITY TECH
FTC TEAM 13684

Odometry

INFINITY TECH
FTC TEAM 13684

Odometry Overview

- **Localization** means the ability to locate the position of the robot at some point in time - especially useful for **autonomous** driving.
- **Odometry** is one type of localization that uses data from encoders.
- The approach described here is based on the **Road Runner** motion planning library.



INFINITY TECH
FTC TEAM 13684

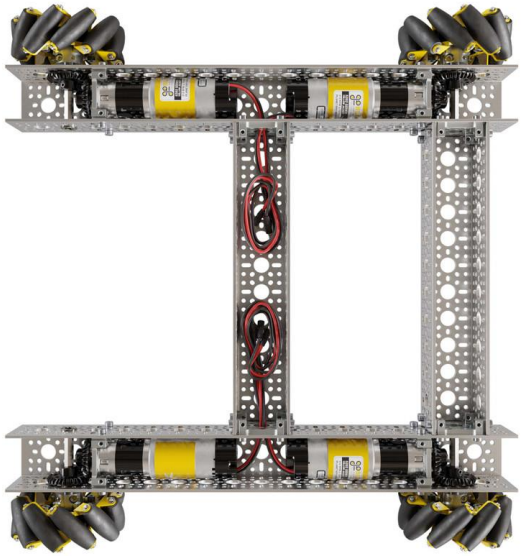
Drive Encoder Localization

Localization can technically be achieved using the encoders on the drive motors

But…

- Mecanum wheel suffer from slippage, which leads to drift
- built-in encoders are not as precise

Odometry pods are the recommended approach



INFINITY TECH
FTC TEAM 13684

Odometry Pods

REV Through-Bore Encoder

8,192 counts per revolution!

Omni-Directional Wheels

small unpowered wheel with rollers to track the distance the robot has traveled through the encoder attached to the wheel's axle.

Odometry Pods

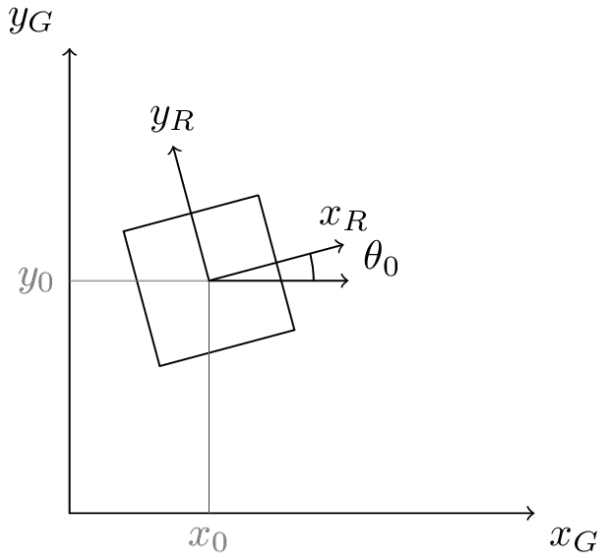
Complete odometry package with encoder, wheels, housing, spring.

INFINITY TECH
FTC TEAM 13684

Pose

- **Pose** specifies the robot position and heading
- **Position** given by **x** and **y** Cartesian coords.
- **Heading** is given by angle Θ that represents direction that the robots front is facing.

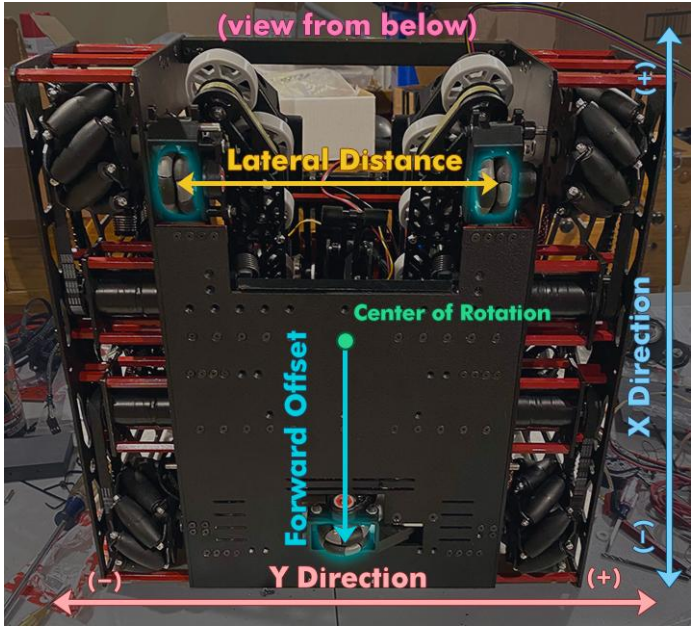
Pose = (x , y, Θ)



INFINITY TECH
FTC TEAM 13684

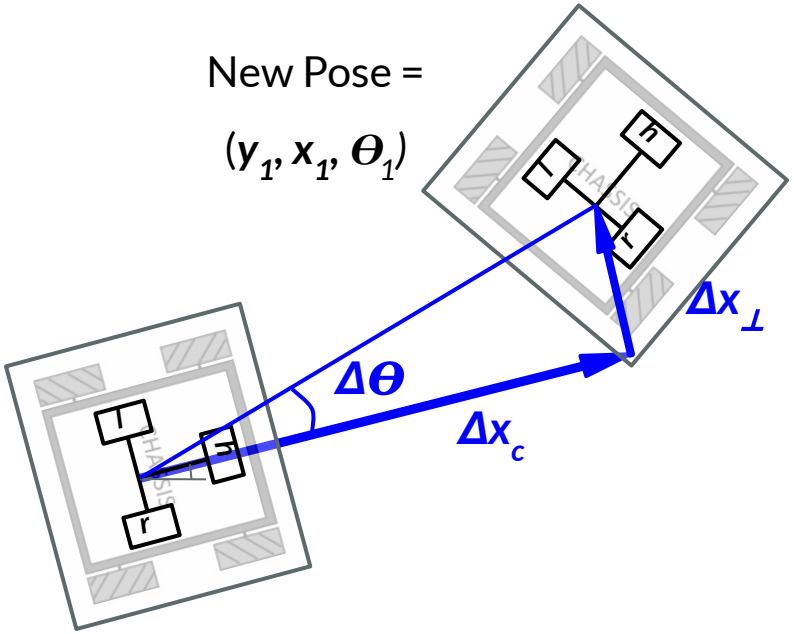
Three-Wheel Odometry Setup

- **Two** encoders that **parallel** with drive wheels
  - Left and right encoders
  - Separated by **lateral distance**, ***L***
- **One** encoder **perpendicular** to drive wheels
  - **Forward offset, *F*,** from center of rotation



INFINITY TECH
FTC TEAM 13684

Odometry Math



Center Displacement

$\Delta x_c = \Delta x_l + \Delta x_r / 2$

Heading Displacement

$\Delta\Theta = \Delta x_l - \Delta x_r / L$

Horizontal Displacement

$\Delta x_\perp = \Delta x_h - (F * \Delta\Theta)$

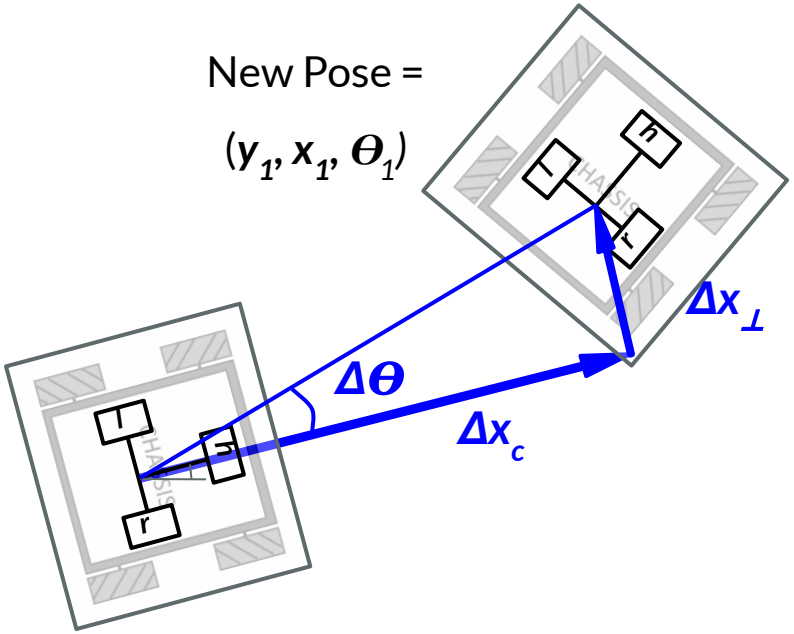
Δx_l, Δx_r, Δx_h *are encoder displacements*

L *is lateral dist between l and r*

F *is forward offset*

INFINITY TECH
FTC TEAM 13684

Odometry Math (cont'd)



$$\Delta x = \Delta x_c \cos\theta_0 - \Delta x_\perp \sin\theta_0$$

$$\Delta y = \Delta x_c \sin\theta_0 + \Delta x_\perp \cos\theta_0$$

Change in Pose

$$x_1 = x_0 + \Delta x$$

$$y_1 = y_0 + \Delta y$$

$$\theta_1 = \theta_0 + \Delta\theta$$

Odometry Pseudocode

```
delta_left_encoder_pos = left_encoder_pos - prev_left_encoder_pos
delta_right_encoder_pos = right_encoder_pos - prev_right_encoder_pos
delta_center_encoder_pos = center_encoder_pos - prev_center_encoder_pos

delta_theta = (delta_left_encoder_pos - delta_right_encoder_pos) / lateral_dist
delta_middle_pos = (delta_left_encoder_pos + delta_right_encoder_pos) / 2
delta_perp_pos = delta_center_encoder_pos - forward_offset * delta_theta

delta_x = delta_middle_pos * cos(heading) - delta_perp_pos * sin(heading)
delta_y = delta_middle_pos * sin(heading) + delta_perp_pos * cos(heading)

x_pos += delta_x
y_pos += delta_y
heading += delta_theta

prev_left_encoder_pos = left_encoder_pos
prev_right_encoder_pos = right_encoder_pos
prev_center_encoder_pos = center_encoder_pos
```

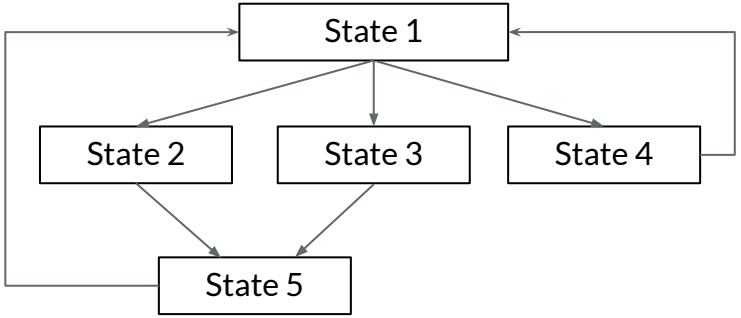
INFINITY TECH
FTC TEAM 13684

Finite State Machine

INFINITY TECH
FTC TEAM 13684

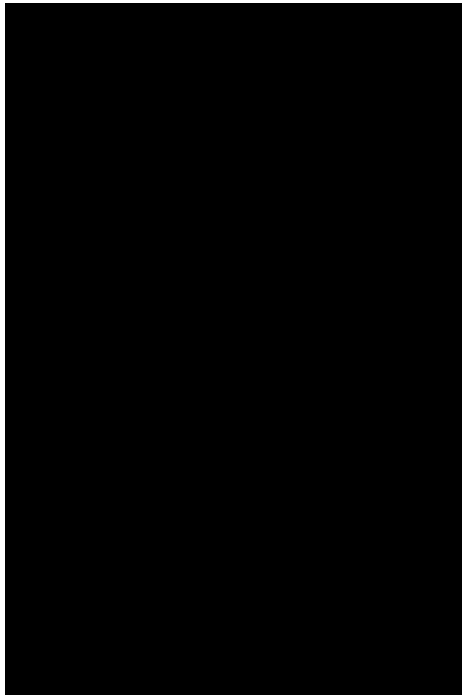
Finite State Machine (FSM)

- FSM is a computational model that can be implemented using software
- the robot or mechanism can be in one of a **finite** number of states at any given time
- can **transition** to a different state once something happens
- allows for tasks to depend on each other's execution in a **non-linear fashion**



Color Sorter Example

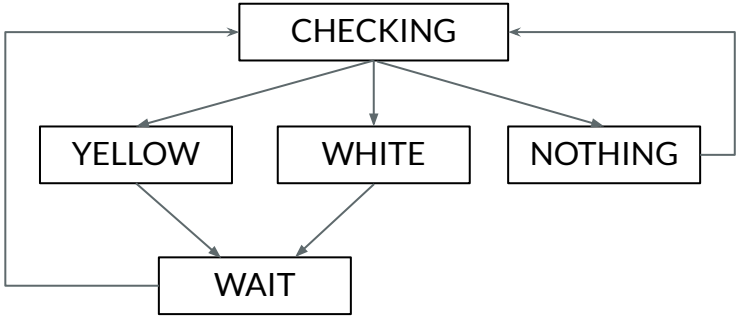
- Infinity Tech's summer project
- Goal was to sort objects depending on their color
- Simply detecting the object and then sending power signal to motor did not work because program loop runs at a 50 Hz rate
- Using `sleep(1000);` is not ideal because entire robot will be idle
- Solution was to use **states**



INFINITY TECH
FTC TEAM 13684

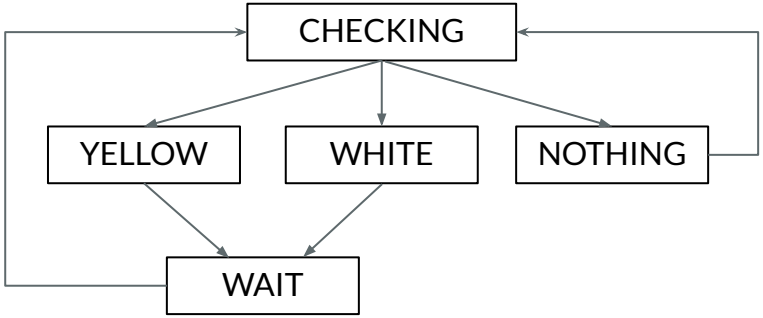
States for Color Sorter

CHECKING	uses color sensor to detect object
YELLOW	rotates sorter CCW if yellow is detected
WHITE	rotates sorter CW if white is detected
NOTHING	does not rotate sorter
WAIT	holding state until rotation is done



Color Sorter Pseudocode

```
switch (currentState) {
    case CHECKING:
        if ("Nothing is detected") {currentState = state.NOTHING;}
        else if ("Yellow is detected") {currentState = state.YELLOW;}
        else if ("White is detected") {currentState = state.WHITE;}
        break;
    case YELLOW:
        // Rotate Motor CCW
        currentState = state.WAIT;
        break;
    case WHITE:
        // Rotate Motor CW
        currentState = state.WAIT;
        break;
    case NOTHING:
        currentState = state.CHECKING;
        break;
    case WAIT:
        // Timer until rotation is complete
        currentState = state.CHECKING;
        break;
}
```



INFINITY TECH
FTC TEAM 13684

Acknowledgements

- Game Manual 0
  - gm0.org
  - Great resource for FTC teams
- FIRST Tech Challenge Docs
  - Ftc-docs.firstinspires.org
- Road Runner
  - learnroadrunner.com

INFINITY TECH
FTC TEAM 13684