



Detailed Design Review  
January 2018



# **Swerve Robotic Platform**

## ***Software and Electronics***

**Prepared for:** Dr. Ajmal Yousuff  
Dr. Tein-Min Tan  
Josh Geating  
Eric Vance  
Shane Simpson  
Chris Noveral

**Prepared by:** Frederick Wachter  
Alexander Nhan

# Report Outline

## Report Outline

1. *Software, Controls, and Electrical Overview*
2. *Electronic Hardware and Layouts*
3. *Simulations*
4. *Robot Model*
5. *Algorithms*
6. *Controls*
7. *Safety Features*
8. *Testing*
9. *Wrap Up*

# Terminology

**Patron** → *human operator*

**Platform** → *Swerve robot*

**Environment** → *surroundings of the platform including all static and dynamic objects*

# **Software, Controls, and Electrical Overview**

Overview of software, controls, and electrical usage, design, and goals

Software, Controls, and  
Electrical Overview

Electrical Hardware  
and Layouts

Simulations



# Software Needs and Purpose




## Software Needs

- Provide interfaces between all electrical components
- Provide methods of passing and storing information to user and through the platform components

## Additional Useful Criteria

- Provide debugging and visualization tools to user
- Provide offline simulation for algorithm development and simulations
- Provide an environment for rapid algorithm development

## Software Framework Choice

- *Robot Operating Systems (ROS)* 
  - Provides interfaces between sensors, computers, and microcontrollers
  - Provides message passing, recording, and playback
  - Provides debugging and visualization tools
- *Gazebo Robot Simulator* 
  - Provides offline robot simulations
  - Provides communication interfaces
- *MATLAB and Robot Systems Toolbox* 
  - Provides rapid development environment
  - Provides communication interfaces
  - Provides easy to use visualization and debugging tools

# Software Tool Features



## Robot Operating System (ROS) Features

- Message passing information
- Recording and playback of messages
- Remote procedure calls
- Distributed Parameter System
- Standard Robot Messages
- Robot Geometric Library
- Robot Description Language
- Diagnostics Tools
- Integrated Robotics Algorithms
- Visualization tools
- Custom Graphical User Interfaces
- Command Line Tools
- Open source libraries and sensor interfaces



## Gazebo Robot Simulator Features

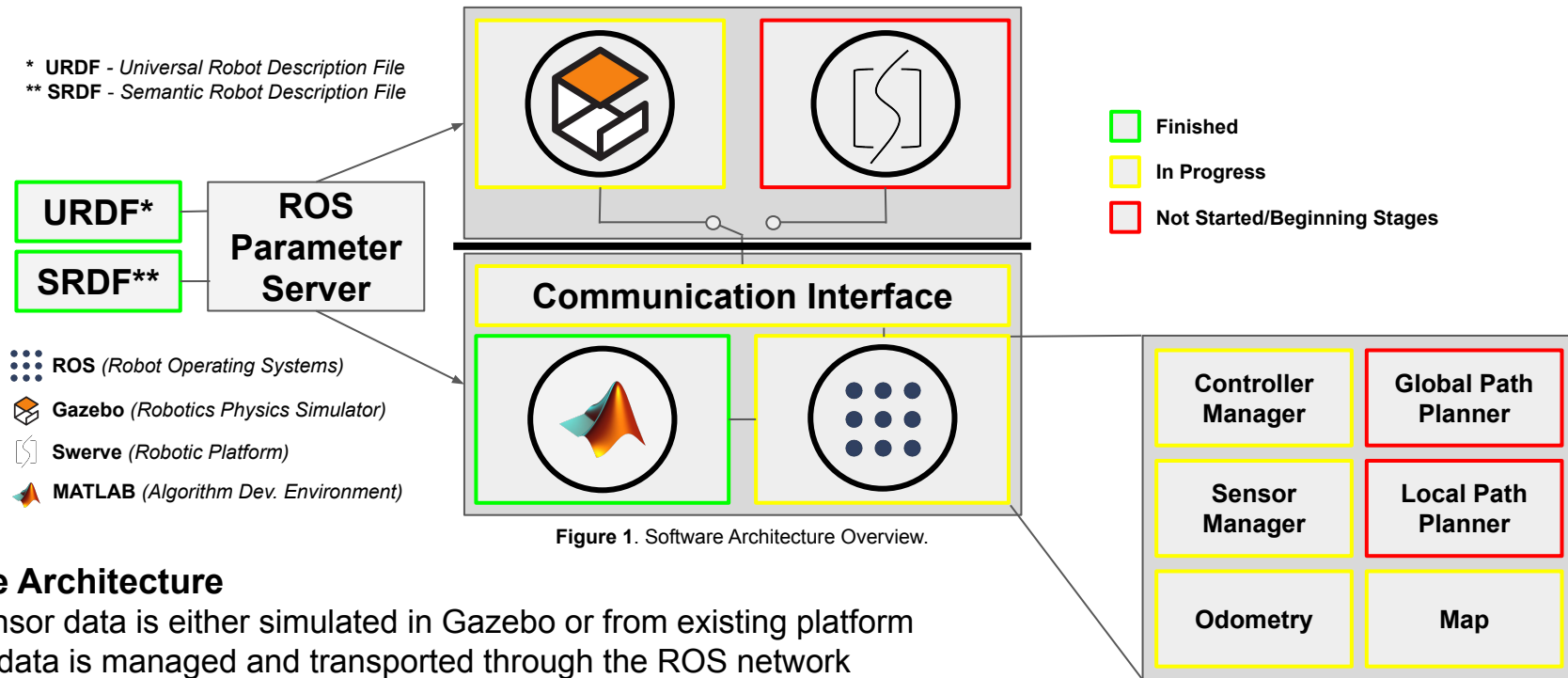
- Dynamics simulations with physics engines
- Advanced 3D graphics
- Simulated sensors with optional noise modeling
- Plugins for robot, sensor, and environment control
- Robot models
- Communication Interfaces
- Cloud simulations
- Command Line Tools



## MATLAB and Robot Systems Toolbox Features

- Provides communication interface with ROS
- Provides robotics algorithms
- Provides rapid development environment

# Software Architecture Overview



## Software Architecture

- Sensor data is either simulated in Gazebo or from existing platform
- All data is managed and transported through the ROS network
- MATLAB can interface with the ROS network to retrieve and send data
- ROS nodes execute the finalized autonomy algorithms and managers

# Controls and Electrical Needs and Purpose

## Controls Needs

- Provide the platform with the ability to drive in a desired direction at a desired rate
  - Be able to achieve high accelerations and speeds
- Provide safety features to prevent misuse of the platform and human safety

## Additional Useful Criteria

- Provide autonomy to the platform for automated operation
- Provide ability to understand the local environment and the platforms position within the environment

## Control and Electrical Solutions

- *Brushless DC Motors*
  - Provides ability to move platform
  - Provides high torque output to achieve high accelerations and speeds
- *Sensors and Controllers*
  - Provides ability to move platform in a desired direction at a desired rate
  - Provides ability to integrate safety features through software and hardware
  - Provides for the platform to describe the local environment
- *Autonomy Algorithms*
  - Provides ability for the platform to understand the environment and location within the environment



# Software Release Schedule

## Software Release Schedule (Releases done on GitHub)

- Release V1.0 (January) - *Simulated Swerve in Gazebo and have all sensor data available in ROS*
- Release V2.0 (March) - *Elemental autonomy algorithms integrated within ROS*
- Release V3.0 (April) - *Data gathering and visualizations from real Swerve platform*
- Release V4.0 (June) - *Finalized prototype autonomy software for offline and online Swerve*

## Swerve Software Links

- Workspace setup and resources: [https://github.com/SwerveRoboticSystems/swerve\\_resources](https://github.com/SwerveRoboticSystems/swerve_resources)
- Swerve ROS package: <https://github.com/SwerveRoboticSystems/swerve>
- Github Organization Page: <https://github.com/SwerveRoboticSystems>

# Electrical Hardware and Layout

Overview of electrical hardware usage and layouts

Software, Controls, and  
Electrical Overview

Electrical Hardware  
and Layout

Simulations

Robot Model



# Electrical Hardware and Simplified Layout



Alien Systems BLDC



VESC BLDC ESC



Teensy CAN Bus Module



Teensy 3.6



AS5047 Encoder



SICK TiM561 LIDAR



Ublox GPS Module



Jetson TX2



ZED Camera



Hall Effect Sensor



Figure 2. Electronics Simplified Layout.

# Calibration and Startup Procedure

## Initial Platform Calibration

- Calibrate all motors with their VESC
- Tune PID parameters for all motors and save configuration file

## Calibration Process for Each Use

- Calibrate encoder position
- Calibrate human weight if in operator mode

## Startup Procedure

- Perform Calibration
- Startup all ROS nodes for hardware communication and backend processes
- Signal to operator platform is calibrated and ready (LED's and GUI)

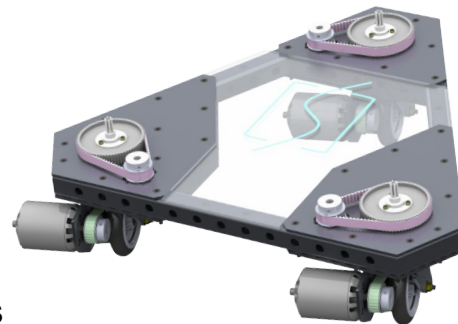


Figure 3. Platform Rendering.

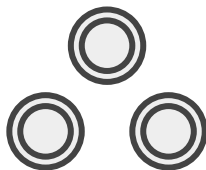


Figure 4. Human-Machine Interface Layout.

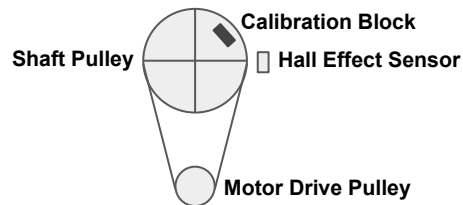


Figure 5. Encoder Calibration.

# Electronics Specifications

## Sensor Hardware Specifications



**SICK TiM561 LIDAR**

**Response Time:** 0.067s (67ms)  
**Scan Frequency:** 15 Hz  
**Depth Range:** 10m  
**Scanning Angle:** 270 degrees  
**Statistical Error:** 0.02m (20mm)  
**Assumed Distribution:** Gaussian (0, 0.00667)



**ZED Camera**

**Scan Frequency:** 100 Hz  
**Odometry:** Real-time depth-based VI SLAM  
**Field of View:** 110 degrees  
**6DOF Orientation Accuracy:** 0.1 degrees  
**6DOF Pose Accuracy:** 0.001m (1mm)



**Ublox GPS Module**

**Time to First Fix:** 26s  
**Update Rate:** 5 Hz  
**Horizontal Accuracy:** 2.5m  
**Heading Accuracy:** 0.3 degrees



**AS5047 Encoder**

**Resolution:** 14 bit  
**Operating Voltage:** 3.3V, 5V compliant  
**Temperature Range:** -40°C to 150°C



**Hall Effect Sensor**

**Operating Voltage:** 2.5V to 24V DC  
**Max Sinking Current:** 50mA  
**Low Current Consumption**

## Computing Hardware Specifications



**Jetson TX2**

**Processor Type:** ARM  
**Memory:** 8GB 126 bit LPDDR4  
**GPU:** 256 CUDA cores  
**CPU:** Quad Arm up to 1.3 GHz  
**Pinouts:** CAN, UART, SPI, I2C, I2S, GPIOs



**Teensy 3.6**

**Clock Speed:** 180 MHz  
**Processor Type:** ARM  
**CAN Ports:** 2  
**GPIO's:** 32  
**PWM PortS:** 22  
**I2C Ports:** 4

## Motor Hardware Specifications



**Alien Systems BLDC**

**Max Power:** 6,000 W  
**Max Voltage:** 18S (75.6V)  
**Max Current:** 200A



**VESC BLDC ESC**

**Voltage Output/Input:** 8V to 60V  
**Max Current:** 240A  
**Feature:** Integrated 5V BEC  
**Feature:** Regenerative Braking  
**Feature:** Sensored and Sensorless Operation

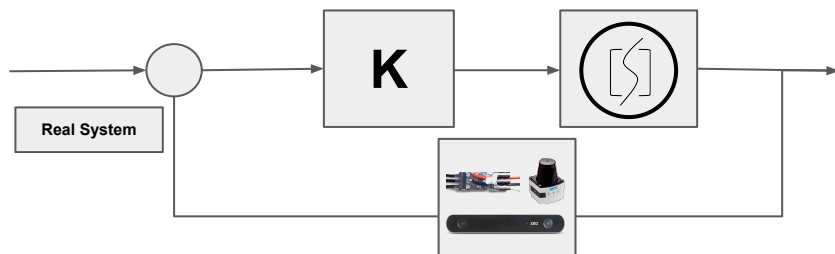
# Simulations

Overview of simulation need, simulated sensors, and Gazebo

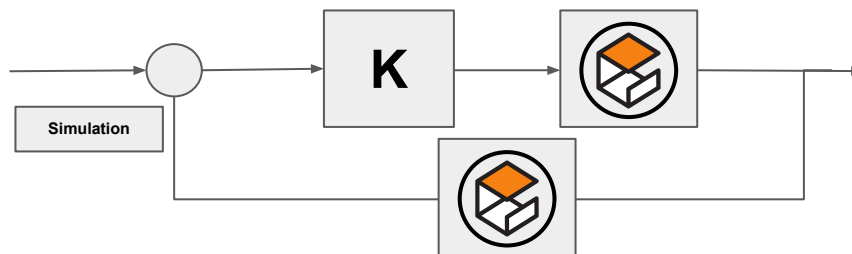
# Simulations Need and Purpose

## Benefits of Offline Simulations

- Allows for testing of controllers and algorithms while the real platform is being built
- Simulates sensor data with optional noise model parameters
- Validation of the dynamic model of the robot
- Algorithms developed using the simulation can be directly implemented onto the real robot
  - Assuming the simulated model is accurate



**Figure 6.** Block Diagram of Control Loop on the Real Platform.



**Figure 7.** Block Diagram of Control Loop on the Simulated Platform.

# Simulated Sensors and Sensor Modeling

## Simulated Sensors Usage

- LIDAR (Light Detection and Ranging) → Simulates SICK TiM561 LIDAR
- Stereo Camera → Simulates ZED Camera stereo camera
- Depth Camera → Simulates ZED Camera depth camera
- IMU (Inertial Measurement Unit) → Need to choose an IMU
- GPS → Simulates Ublox GPS Module

## Error Modeling

- Assumption is made that all sensors can be modeled with Gaussian noise
- Need to test real sensors to determine error model parameters
  - Models will most likely be modeled with a zero mean with a variance that is determined through testing
  - Testing should be done in a Vicon room in order to get an accurate ground truth

$$\hat{z} = z + \mathcal{N}(0, \Sigma)$$

$z \rightarrow \text{ground truth}, \hat{z} \rightarrow \text{sensor output}$

**Equation 1.** Assumed sensor noise model equation.



# Current State of Simulation

## Current State of Gazebo Simulation (as of 2017/12/31)

- Ability to simulate LIDAR and stereo cameras
- Having issues with simulated depth cameras most likely due to GPU driver on computer
- Currently not simulating IMU or GPS data until they are needed

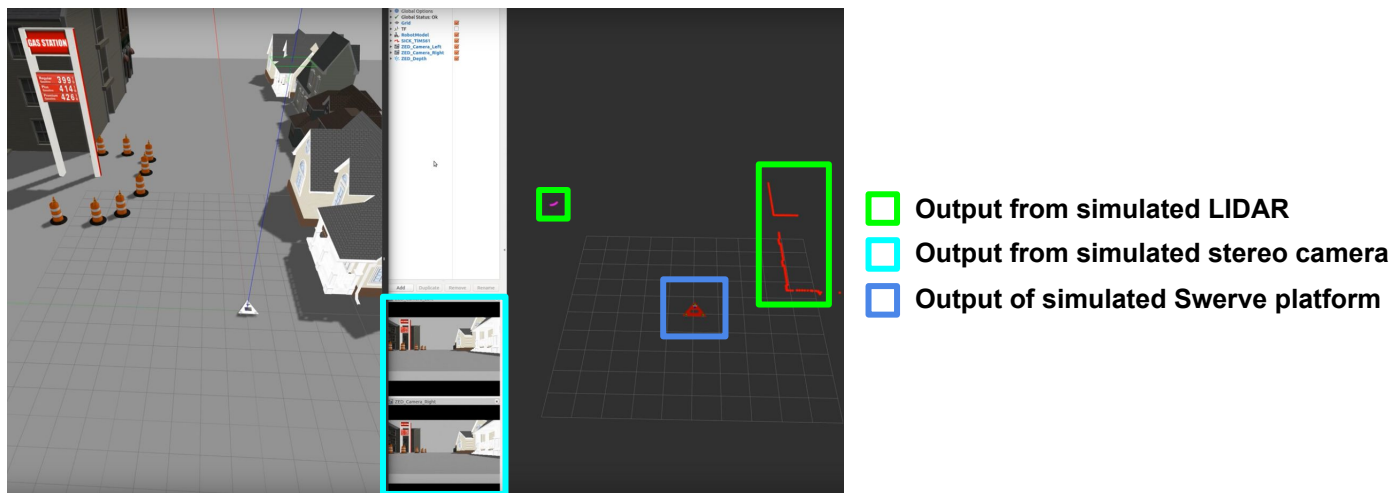


Figure 8. Output from simulated sensors on simulated Swerve platform.

# Robot Model

Robot description and dynamic model

# Robot Description

## Robot Description Overview

- The platform is described through a tree of static and non-static frames
- Non-static tree links are updated using sensor data from simulated or existing platform
- The relationship between the world and odom frames is described by the state estimator
- All sensor data is projected from its frame to the platform body frame for map building and updates
- Frame tree's allow for easy transformations of points in space to be described between different frames

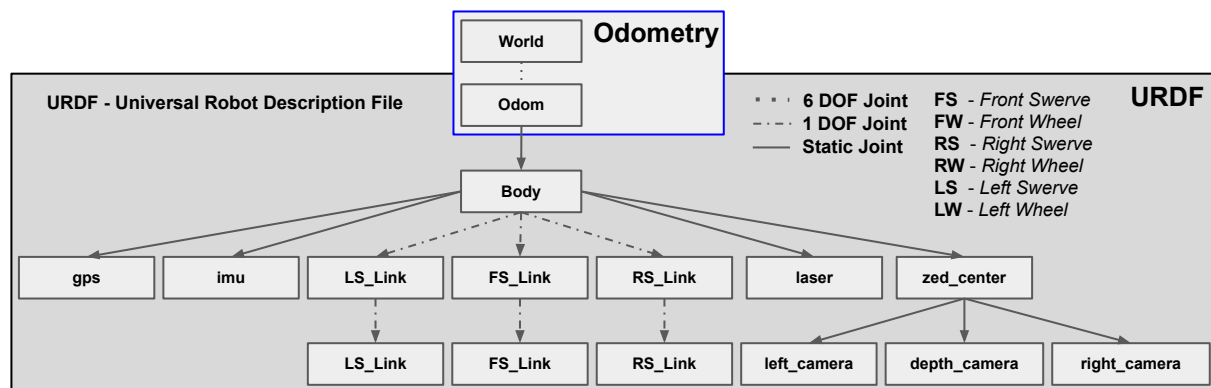


Figure 9. Frames Tree of the Swerve Platform.

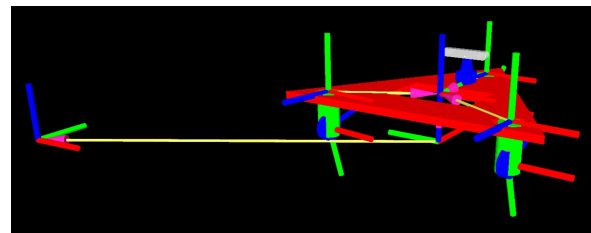


Figure 10. Visual of Transform Tree for Swerve.

# Dynamic Model

## Dynamic Model of Platform

- Used to track the robot's state over time
- Determines the relationship between the states and output parameters
- Determines the state transition matrix to update the robot state between time steps
  - State update function used in Kalman/Particle Filter
- Dynamic model still needs to be developed

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k$$
$$\mathbf{z}_k = \mathbf{C}\mathbf{x}_k$$

**Equation 2.** Swerve Discrete State-Space Model.

# Algorithms

Algorithms used to describe the environment and platform localization



# Odometry

## Odometry

- Used to estimate and track the position of the robot in the local environment
- Estimates of the robot state and finds innovations driving a stochastic process given a set of observations

## Odometry Filter Options

- *Kalman Filter*
  - Recursive linear approximations with a tracked state covariance
- *Particle Filter*
  - Sequential Monte Carlo method using particles that describes the statistics of the system's distribution
- Both use a modified version of the dynamic model to include inherent errors in modeling and sensors
- Both are described in detail on the next two slides

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k + \mathcal{N}(\mathbf{0}, \mathbf{Q})$$

$$\mathbf{z}_k = \mathbf{C}\mathbf{x}_k + \mathcal{N}(\mathbf{0}, \mathbf{R})$$

**Equation 3.** Swerve Discrete State-Space Model with Gaussian Noise.

# Odometry - Kalman Filter

## Kalman Filter Description

- Uses a probabilistic discrete state-space model of the system
- Requires an initial state and state covariance along with system and observation covariances

## Kalman Filter Process

- Uses previous state and covariances to predict the current new state
- Update the Kalman gain
- Update the state with observations based on Kalman gain
- Update the state covariance using the Kalman gain
- Repeat

$$\begin{aligned}\mathbb{P}(\mathbf{x}_{t+1}|\mathbf{x}_t) &= A_k \mathcal{N}(\mathbf{x}_t, \mathbf{P}_t) + \mathcal{N}(\mathbf{0}, \mathbf{Q}_t) \\ \mathbb{P}(\mathbf{z}_t|\mathbf{x}_t) &= C_k \mathcal{N}(\mathbf{x}_t, \mathbf{P}_t) + \mathcal{N}(\mathbf{0}, \mathbf{R}_t) \\ \hat{\mathbf{x}}_t &= \underset{\mathbf{x}_t}{\operatorname{argmax}} \mathbb{P}(\mathbf{x}_t | \mathbf{z}_t \cap \mathbf{x}_{t-1})\end{aligned}$$

**Equation 4.** Swerve Discrete State-Space Probabilistic Model.

**Equation 5.** Kalman Filter Flow Chart.

$$\begin{aligned}\mathbf{K} &\in \mathbb{R}^{n \times m}, \mathbf{P} \in \mathbb{R}^{n \times n}, \mathbf{Q} \in \mathbb{R}^{n \times n}, \mathbf{R} \in \mathbb{R}^{m \times m} \\ \mathbf{x} &\in \mathbb{R}^{n \times 1}, \mathbf{z} \in \mathbb{R}^{m \times 1}, \mathbf{A} \in \mathbb{R}^{n \times n}, \mathbf{B} \in \mathbb{R}^{n \times p}, \mathbf{C} \in \mathbb{R}^{m \times n}\end{aligned}$$

# Odometry - Particle Filter

## Particle Filter Description

- Uses an initial set of particles where the statistics of the particles match the statistics of the state
- Each particle contains its own state and an associated weight representing the probability of the state

## Particle Filter Process

- Sample the distribution of the state into a set of particles with uniform weights
- Update the states of each particle individually based on bounded probability of the state change
- Update weights based on likelihood of state update
- Take an observation
- Update weights based on probability of the particles states based on the observation
- Repeat

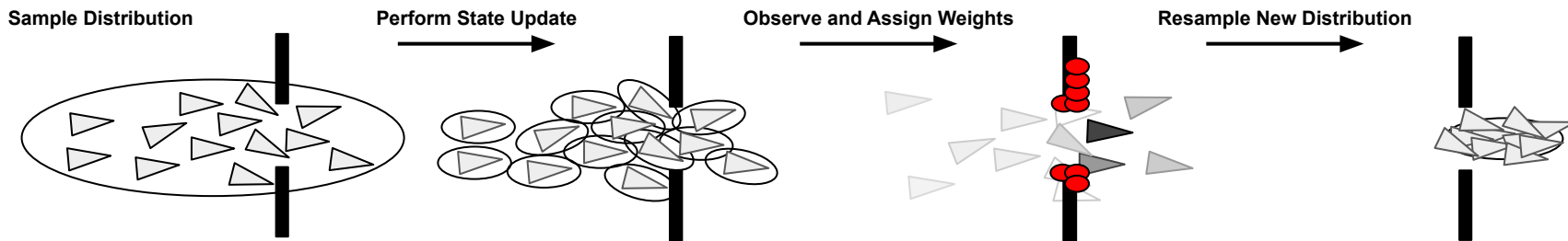


Figure 11. Particle Filter Process Visualization.



# Occupancy Grid

## Occupancy Grid Purpose

- Probabilistic 2D representation of the environment that discretizes environment into cells
- Requires a Bayesian filter to maintain map
- Uses Bayesian probability of cell occupancy combined with measurements to update occupancy cells
- Resulting map can be used to for path planning purposes
- Accuracy of map depends on accuracy of state estimate in order to perform accurate measurement updates

$m_{x,y} : \{1, 0\}, 0 \rightarrow \text{free}, 1 \rightarrow \text{occupied}, m \rightarrow \text{occupancy cell at } (x, y)$   
 $z : \{1, 0\}, 0 \rightarrow \text{free}, 1 \rightarrow \text{occupied}, z \rightarrow \text{observation}$

**Equation 6.** Occupancy Grid Random Variables.

$$\mathbb{P}(m_{x,y}|z) = \frac{\overset{\text{Likelihood}}{\mathbb{P}(z|m_{x,y})} \overset{\text{Prior}}{\mathbb{P}(m_{x,y})}}{\underset{\text{Measurement}}{\mathbb{P}(z)}}$$

**Equation 7.** Bayesian Formulation of Cell Occupancy Probability.

$$\text{odds}(m_{x,y} = 1|z) = \frac{\overset{\text{New Map Value}}{\mathbb{P}(m_{x,y} = 1|z)}}{\mathbb{P}(m_{x,y} = 0|z)} = \frac{\overset{\text{Measurement Update}}{\mathbb{P}(z|m_{x,y} = 1)}}{\mathbb{P}(z|m_{x,y} = 0)} + \frac{\overset{\text{Previous Map Value}}{\mathbb{P}(m_{x,y} = 1)}}{\mathbb{P}(m_{x,y} = 0)}$$

**Equation 8.** Occupancy Grid Update.

# Occupancy Grid Creation Process Visualization

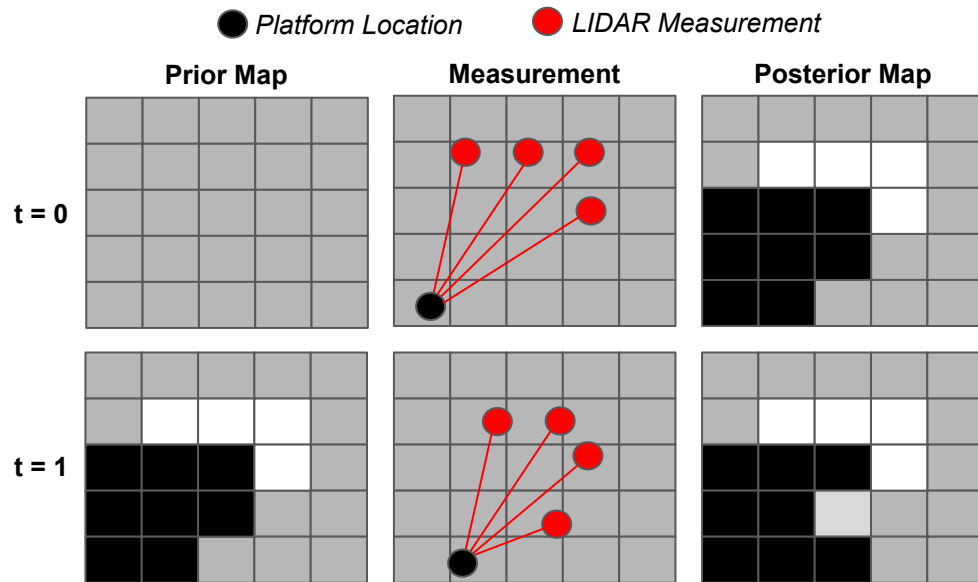


Figure 12. Visualization of Occupancy Grid Generation.

$$odds(m_{x,y} = 1|z) = \frac{\mathbb{P}(m_{x,y} = 1|z)}{\mathbb{P}(m_{x,y} = 0|z)} = \frac{\mathbb{P}(z|m_{x,y} = 1)}{\mathbb{P}(z|m_{x,y} = 0)} + \frac{\mathbb{P}(m_{x,y} = 1)}{\mathbb{P}(m_{x,y} = 0)}$$



Figure 13. Final Occupancy Grid Example.

# Local and Global Planners

## Local and Global Planner Purpose

- Global planners generate initial, valid, non-optimal trajectories from a start to goal location
- Local planners generate recursive paths to avoid collisions due to variations in the map over time

## Global Planners

- RRT (Rapidly-Exploring Random Tree)
- A\* (A-Star)

## Local Planners

- MPC (Model Predictive Controller)

*Work still needs to go into deciding on local and global planner*

# Controls

Control scheme to move platform at a desired direction and rate



# Controller Needs

## Controller Needs

- Provide the controller with desired platform direction and speed
- Controller commands motors to achieve overall platform direction and speed

## Controller Type

- VESC's use an onboard PID controller to control either motor velocity or position

## Possible Control Inputs

- Vector representing the desired platform direction and speed (operator mode)
  - Generated from the human-machine interface, explained on next slide
- A desired end location to move the platform to (autonomous mode)
  - Global planner generates an initial valid trajectory based on current map of the environment
  - Local planner provides controller with discrete set of poses to achieve at specific time intervals



Figure 14. Visual Representation of Operator Mode Control Input.

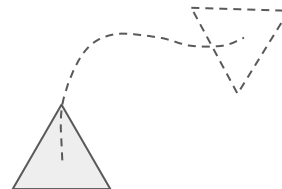


Figure 15. Visual Representation of Autonomous Mode Control Input.

# Operator Mode - Human-Machine Interface

## Interface Overview

- Three sensors at each foot of the patron
- Sensor outputs are triangulated to provide a desired direction
- Amplitude of sensor outputs provides desired rate of platform

## Interface Version 1

- Force sensors as input
- Pro: Already have sensor and easy/simple to integrate
- Con: Does not provide haptic feedback

## Interface Version 2

- Springs with spring potentiometers as input
- Pro: Provides haptic feedback to the user
- Con: Complex platform interface design required
- It will be built if time permitting

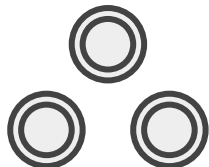


Figure 17. Force Sensor Interface Layout.



Figure 18. Spring Interface Layout.



Figure 16. Swerve Platform with Patron.

# Autonomous Mode

## Autonomy Scheme

- Goal location provided by user or by autonomy manager
- Global planner provides rough trajectory based on map
- Local planner provides trajectory to a receding horizon
  - Local planner performs obstacle avoidance
- Local trajectory is discretized into desired poses
- Controller attempts to move robot to desired pose
  - Goal time between reaching poses is fixed
  - Distances between current pose and desired pose determines speed of platform
  - Velocities of each motor is determined by kinematically projecting the platform velocity down to each motor
  - Each motor operates on its own control loop

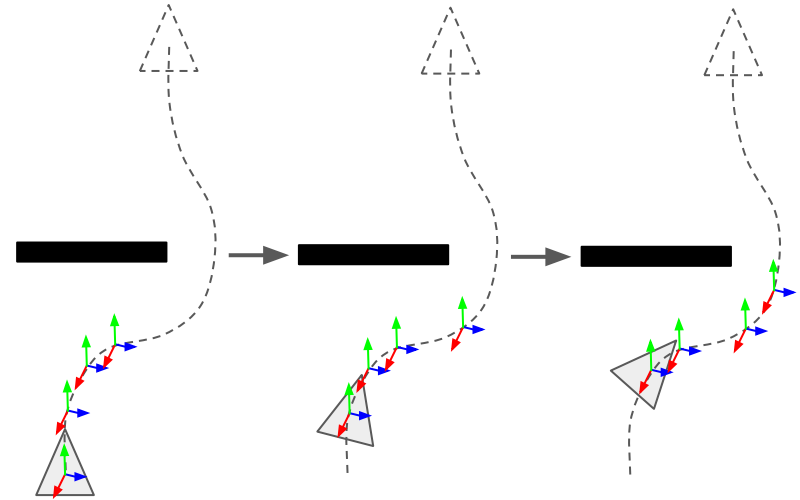


Figure 19. Visual Representation of Autonomy Scheme.

# **Safety Features**

Safety features for the operator and autonomous mode





# Safety Features and Requirements

## Safety Requirements

- Ensure the safety of the patron in operator mode
- Ensure the safety of objects within the environment of the platform in both operator and autonomous mode

## Safety Features

- Patrons must wear ignition lanyards to ensure platform knows if the rider is thrown off
- Provide the possibility to limit max velocity and acceleration
- Enforce a virtual bubble around the platform where it will e-brake when objects are within the bubble
- Shield all exposed wire to avoid damage from external sources

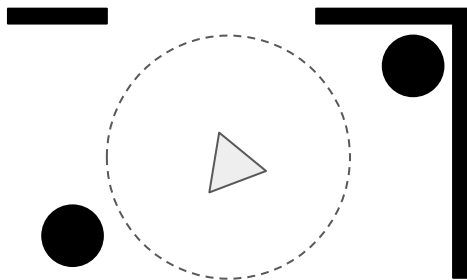


Figure 20. Virtual Bubble Visualization.



Figure 21. Safety Ignition Lanyard.

# Testing

Required electrical and algorithm testing



# Error Modeling Tests

## Error Modeling Need

- Need an understanding of fundamental errors in sensor and model in order to track the platform accurately
- All sensors need to be tested in order to determine bounded accuracy of measurements
- Dynamic model needs to be tested in order to determine drift when using strictly the dynamic model

## Testing Method

- Vicon motion capture systems provide a highly accurate ground truth
- Trackers are placed on the platform, sensors, and objects in the environment
- Locations of all devices and objects are saved in a time synchronous fashion
- Error model parameters are determined by playing back saved locations of devices and objects over time

## Availability of Testing System

- University of Pennsylvania has some Vicon rooms, we will search first at Drexel

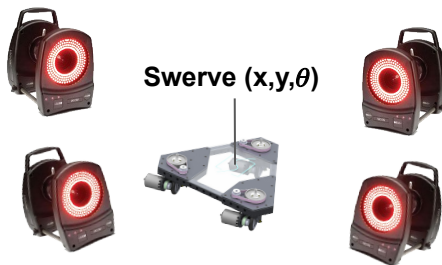


Figure 22. Vicon Test Setup Example.

# Algorithms and Controls Testing

## Kalman vs Particle Filter

- Need to test accuracy and computational load of both filters

## Occupancy Grid

- Need to test to determine if this is sufficient for operating in dynamic environments
- Need to determine computation load of algorithm

## Autonomy and Controls Scheme

- Need to determine if the autonomy and control scheme is stable and can operate with all expected inputs

## Calibration

- Determine threshold for calibrating encoders

## Platform Capabilities

- Test the max velocity and acceleration of the platform
- Test maneuverability of the platform

# Wrap Up

Current state and knowledge gaps



# Wrap Up

## Current State

- Simulated Swerve in Gazebo, need to add additional simulated sensors
  - Official software release will be made in a few weeks once finished
- Initial work done with sensors and electrical components
  - Testing motor, encoder and VESC configuration
  - Integrated LIDAR and worked with features of Teensy 3.6 and Jetson
- Finished course on Estimation and Learning by University of Pennsylvania on Coursera (online)

## Knowledge Gaps

- State Estimation
- Motion Planning
- Dynamics Development for Control

## Resources

- Robotics Specialization - University of Pennsylvania - Coursera Online Courses
  - Computational Motion Planning
  - Mobility
  - Estimation and Learning

# Related Links

## Swerve Software Links

- Workspace setup and resources: [https://github.com/SwerveRoboticSystems/swerve\\_resources](https://github.com/SwerveRoboticSystems/swerve_resources)
- Swerve ROS package: <https://github.com/SwerveRoboticSystems/swerve>

## Swerve Online Presence

- Github Organization: <https://github.com/SwerveRoboticSystems>
- Website: <https://SwerveRoboticSystems.github.io/>

# Contact Us



**SwerveRoboticSystems@gmail.com**



**SwerveRoboticSystems.github.io**