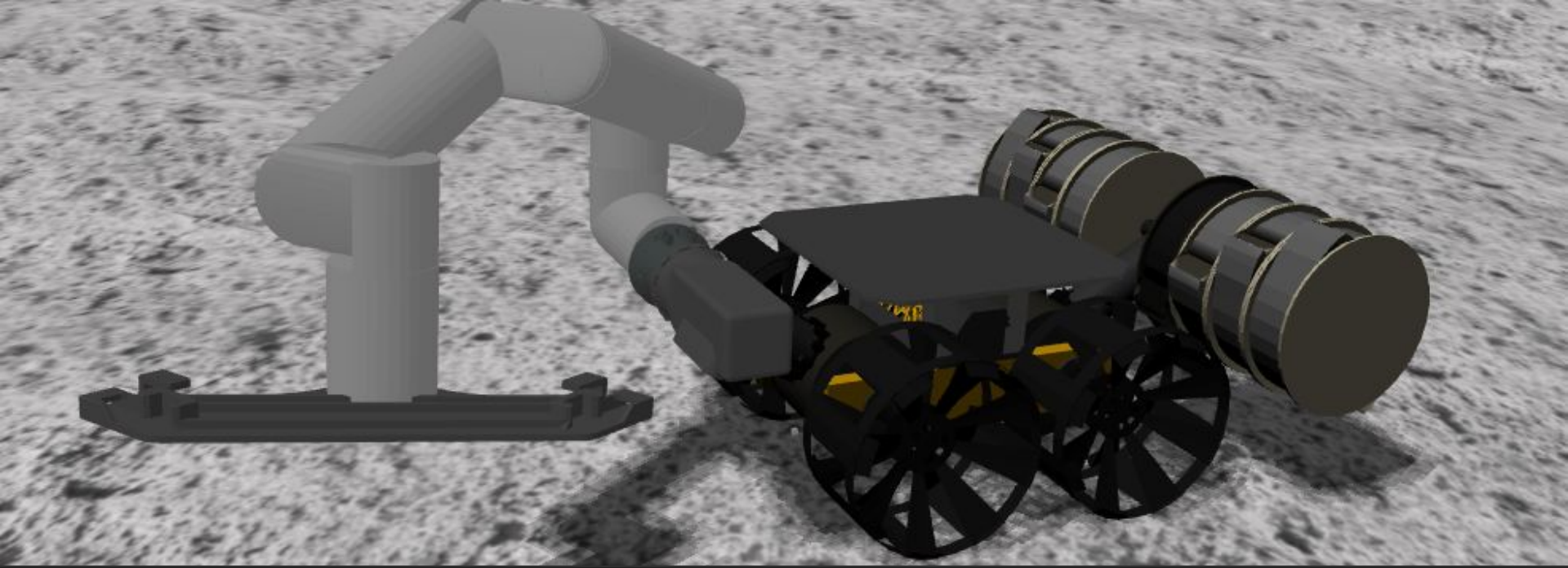


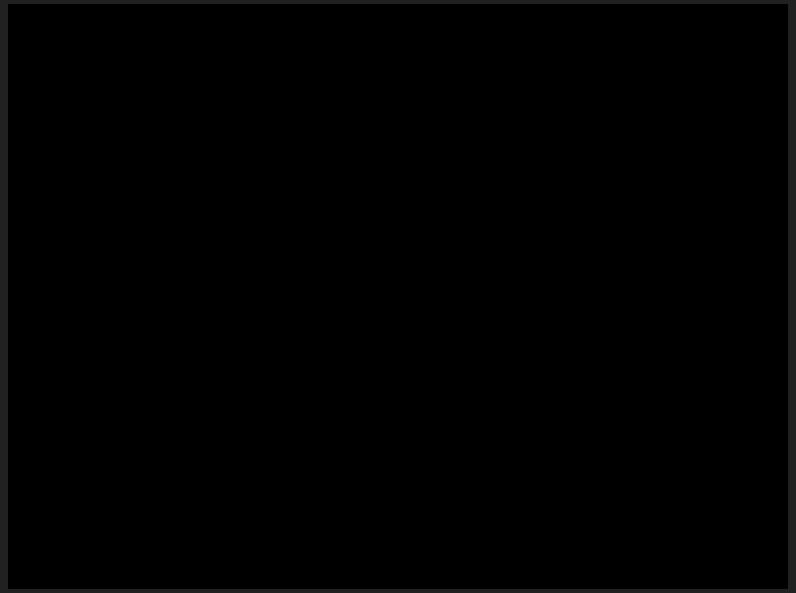
EZ-RASSOR ARM

Team: Austin Dunlop, Cooper Urich, Luca Gigliobianco, Christopher Jackson, and Robert Forristall

Sponsor: Mike Conroy (Florida Space Institute at UCF)

The Project Goal

Build upon the existing EZRASSOR software to add an autonomous functioning arm that uses pavers to build a landing pad on the moon.

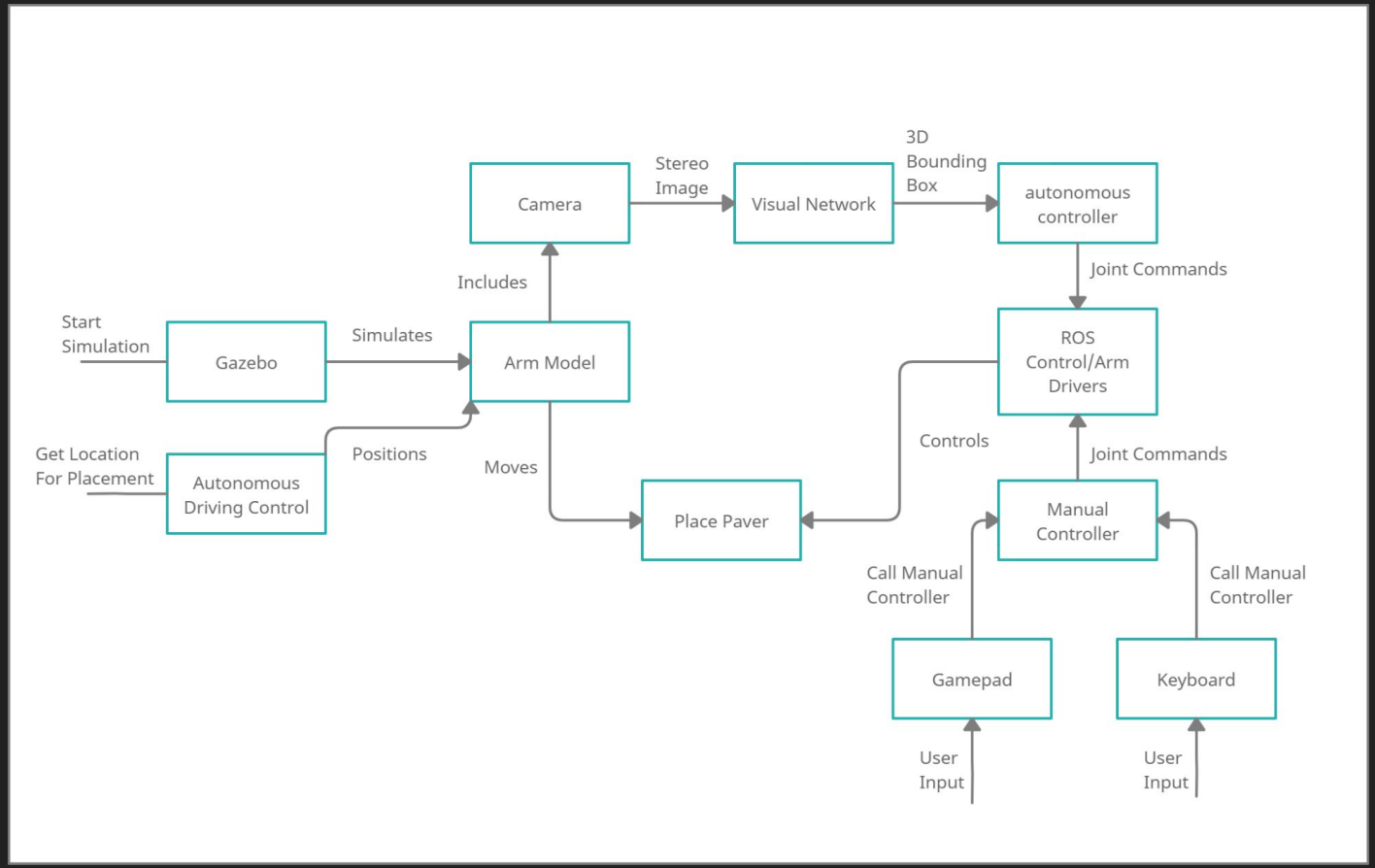


Representation of Arm movement in AutoDesk Fusion 360. Provided by Jacob Bruckmoser.

Requirements

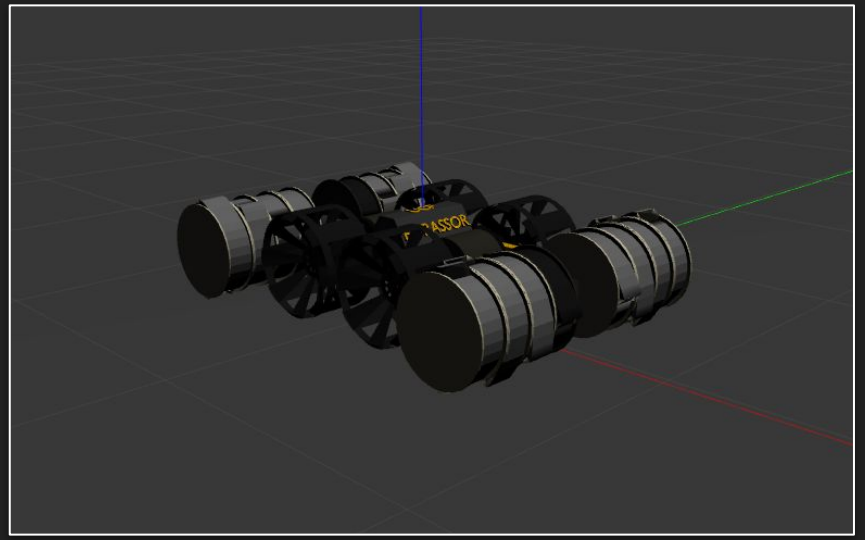
- Required:
  - Add the arm model to the existing Gazebo simulation environment
  - Create controllers for the arm’s motors using ROS
  - Create a visual network that can be used to ensure the placement of the next paver
  - Create an autonomous solution using the visual network and driver controls for the arm’s movements
  - Test solution in the Gazebo simulation environment
- Preferred:
  - Integrate the simulation solution with the real-world arm/rover hardware
  - Add gamepad functionality for the arm using existing rover gamepad app
- Stretch-Goal:
  - Integrate solution with fellow EZ-RASSOR team (Group 28) to allow for swarm controlled paver placement

Block Diagram



Existing EZ-RASSOR

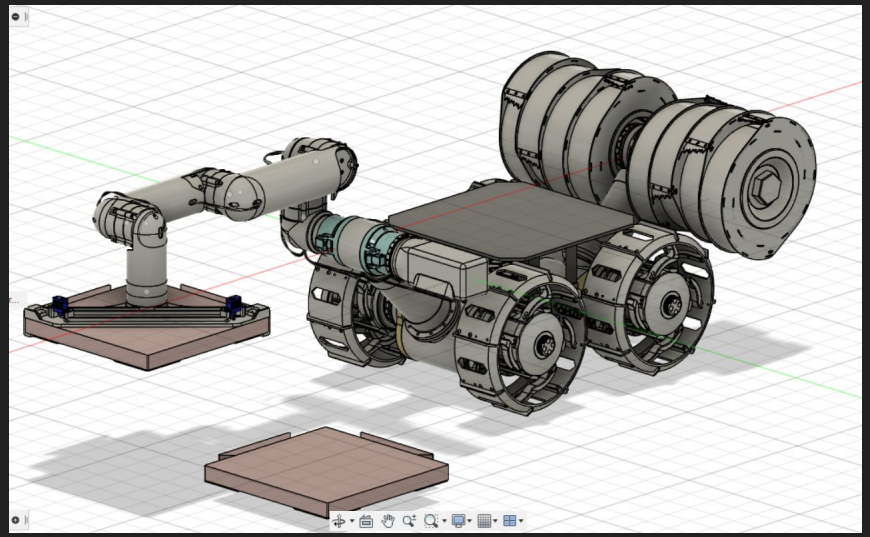
- Rover simulation in Gazebo
  - Autonomous driving control
- Gamepad controls
- Software
  - Ubuntu 18.04 2018
  - Python 2
  - ROS Melodic

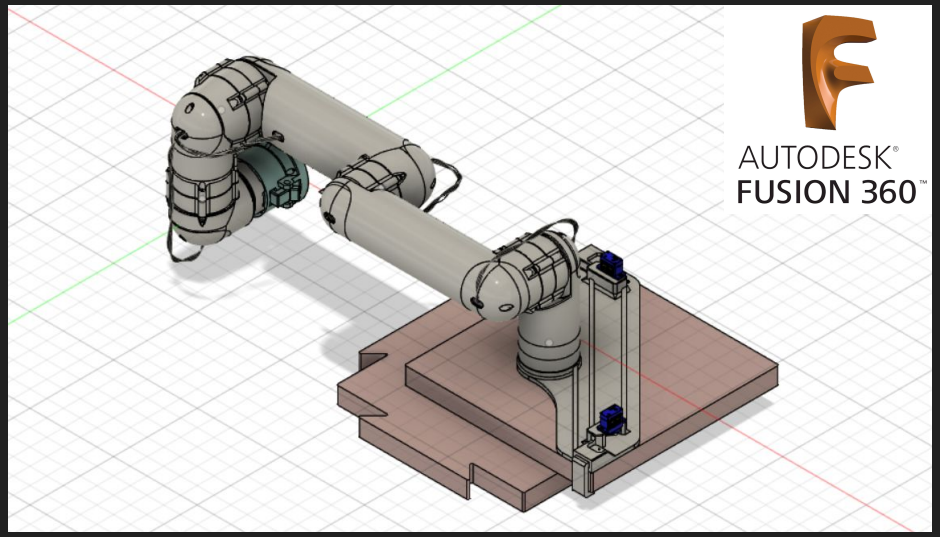


Rover simulation with two Regolith Drums (no Arm).
Inherited from previous EZRASSOR teams.

EZ-RASSOR Arm Design

Arm designed by Jacob Bruckmoser

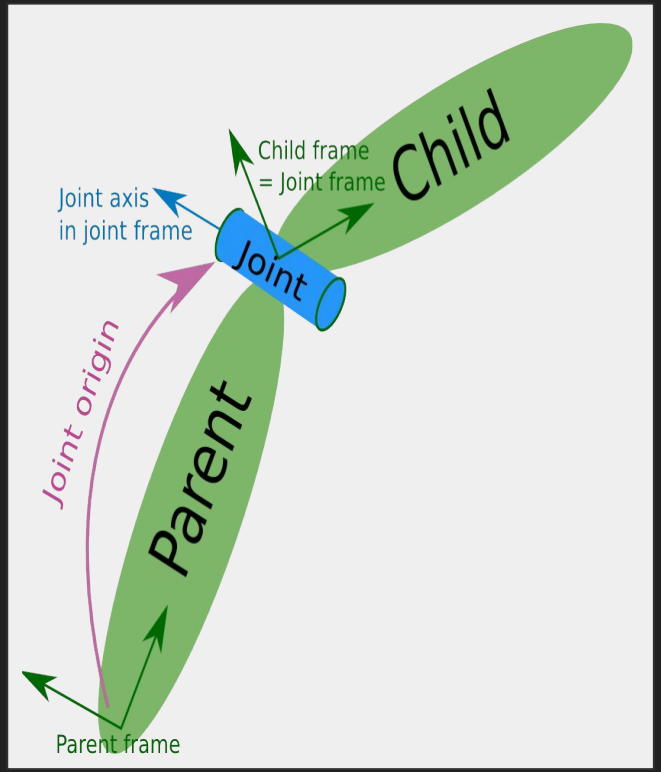




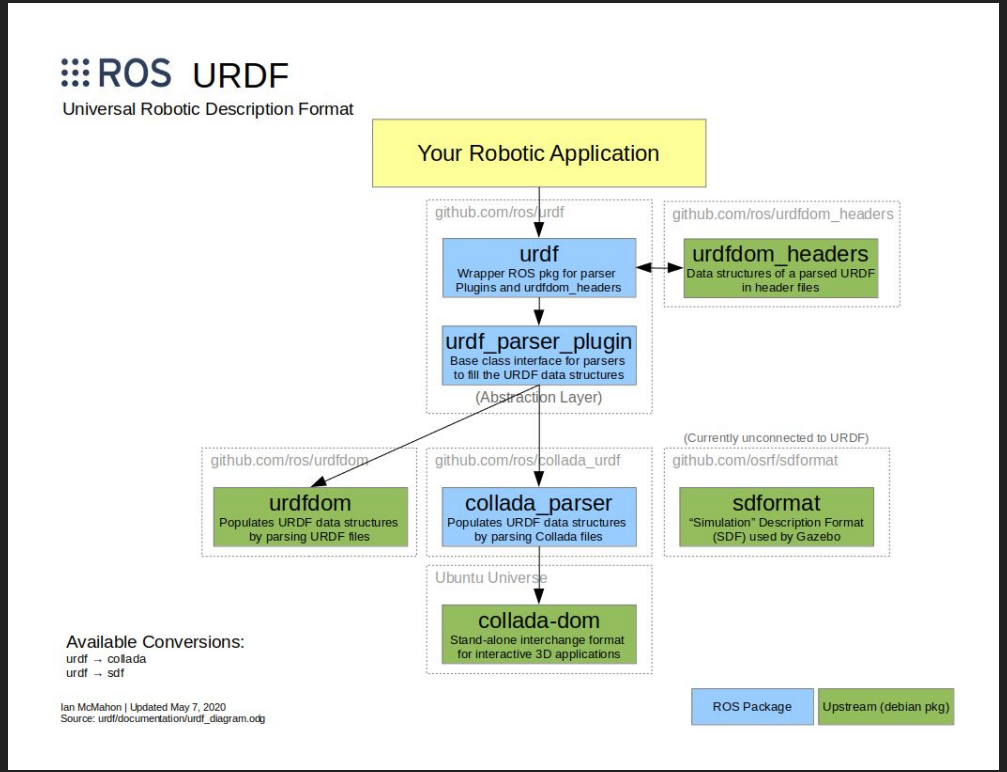
AutoDesk Render of Rover with Arm. Provided by Jacob.

ROS (Robot Operating System)

ROS.org

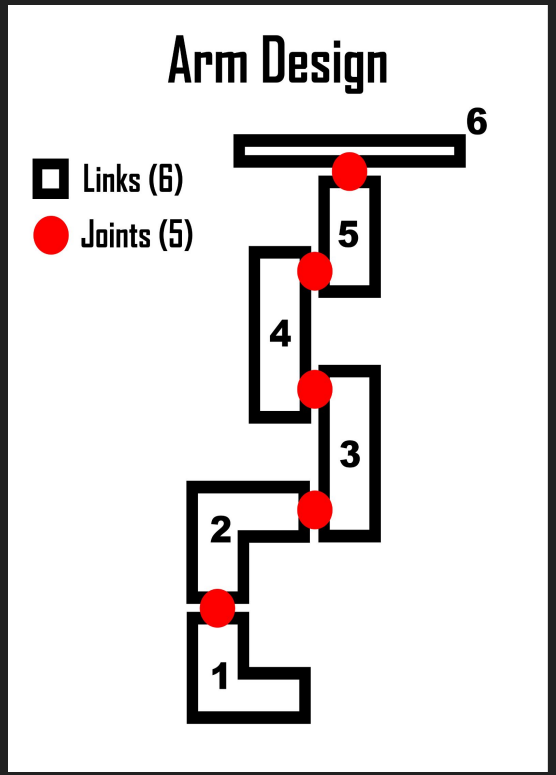


Visual of ROS robot joint and link relationships

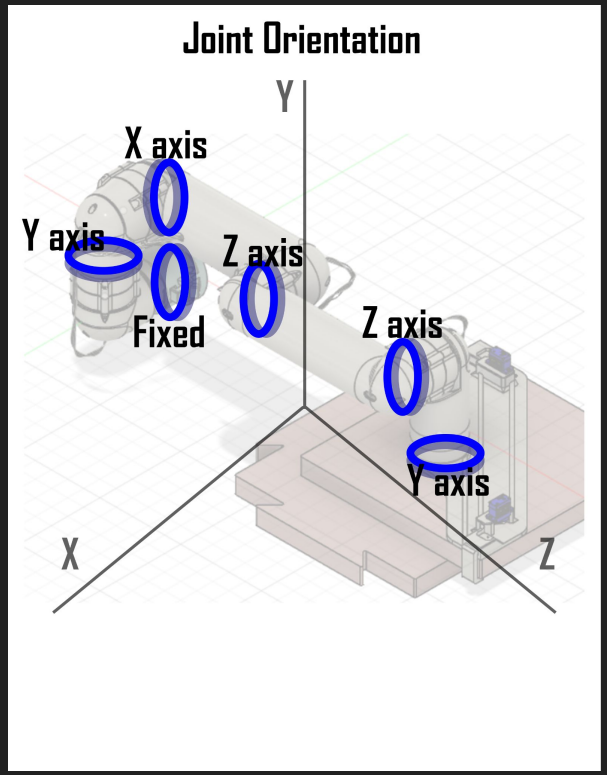


Relationship between packages and components that make up URDF

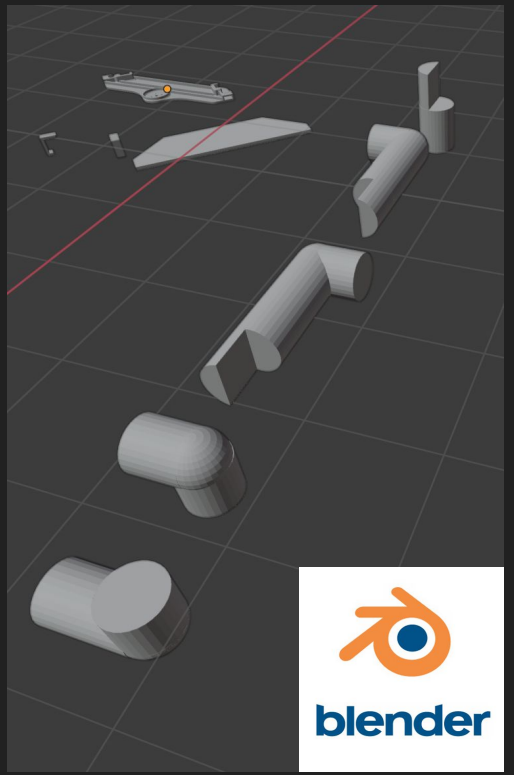
Design Process



Visual of Arm links and joints to be implemented in URDF

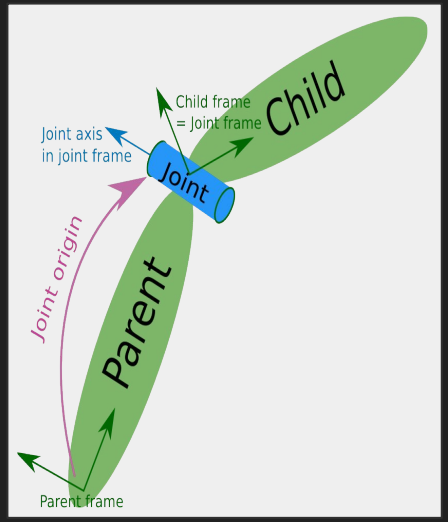


Visual of Joint Orientation For URDF

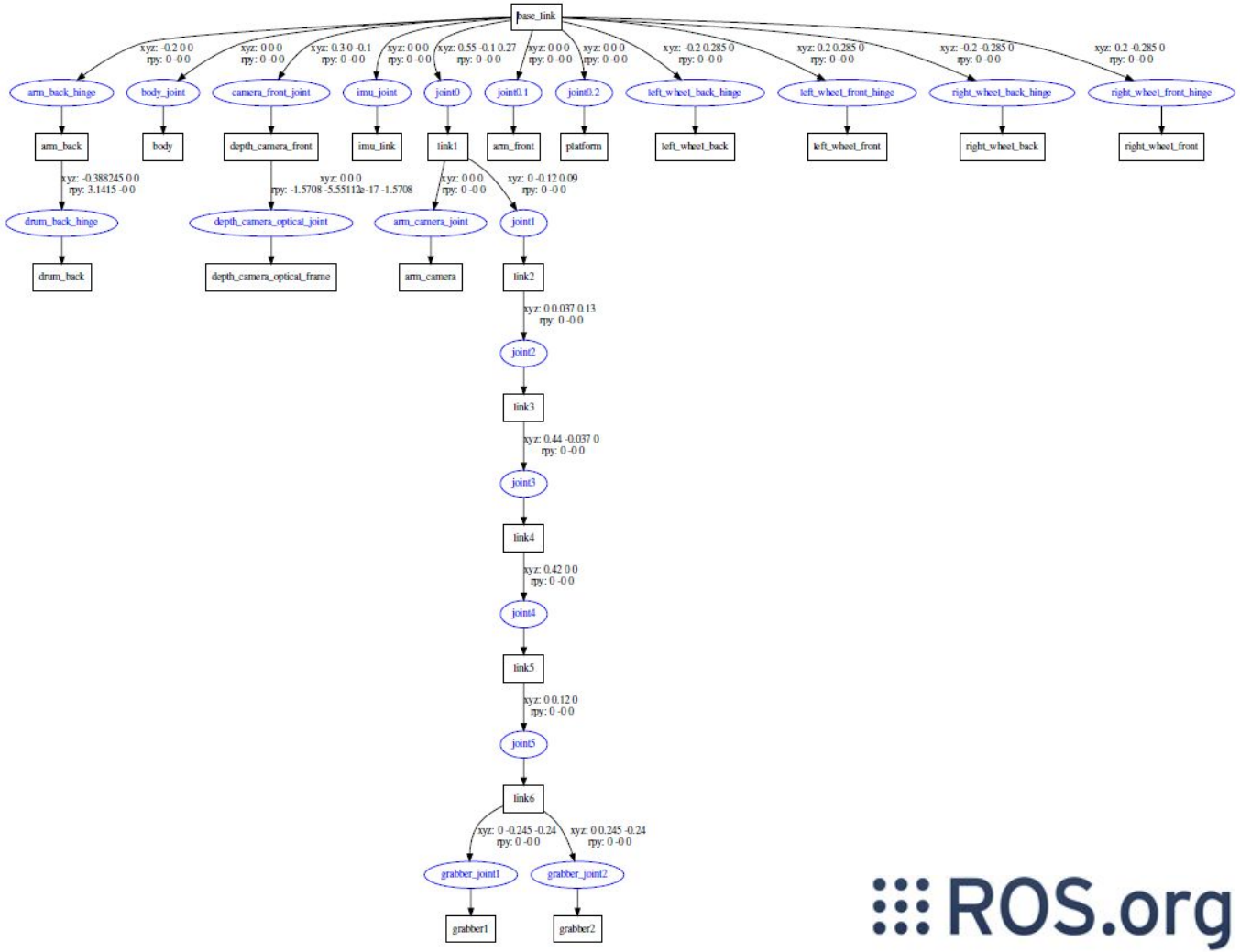


Arm links created and displayed in Blender

Structure of URDF file architecture. Visual created with graphiz2 (right).

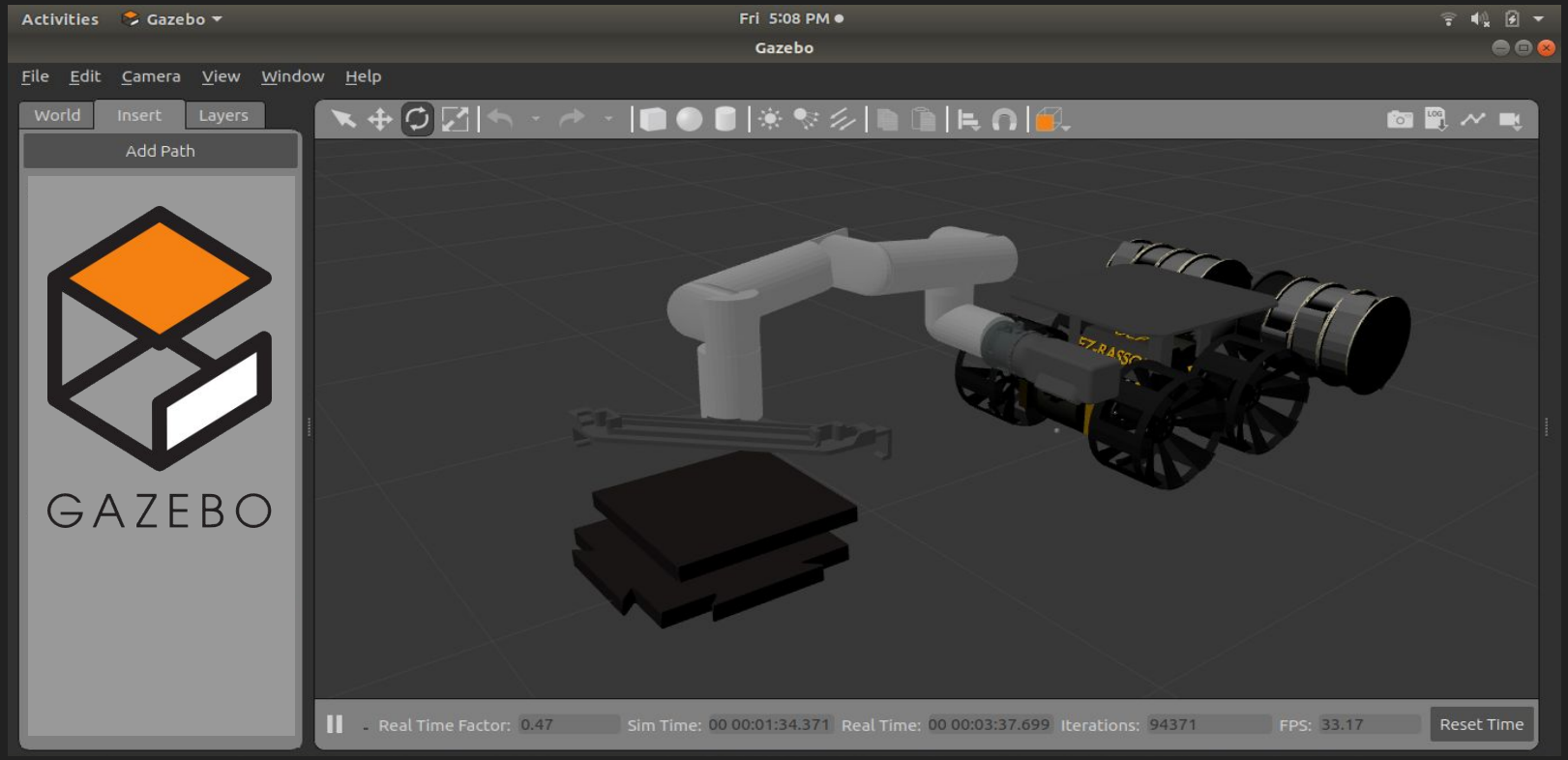


Visual of ROS robot joint and link relationships (above).



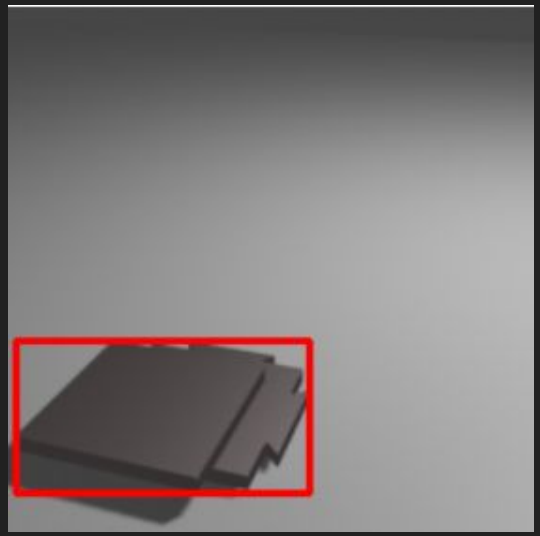
ROS.org

Final Simulation with Arm and Camera added in Gazebo



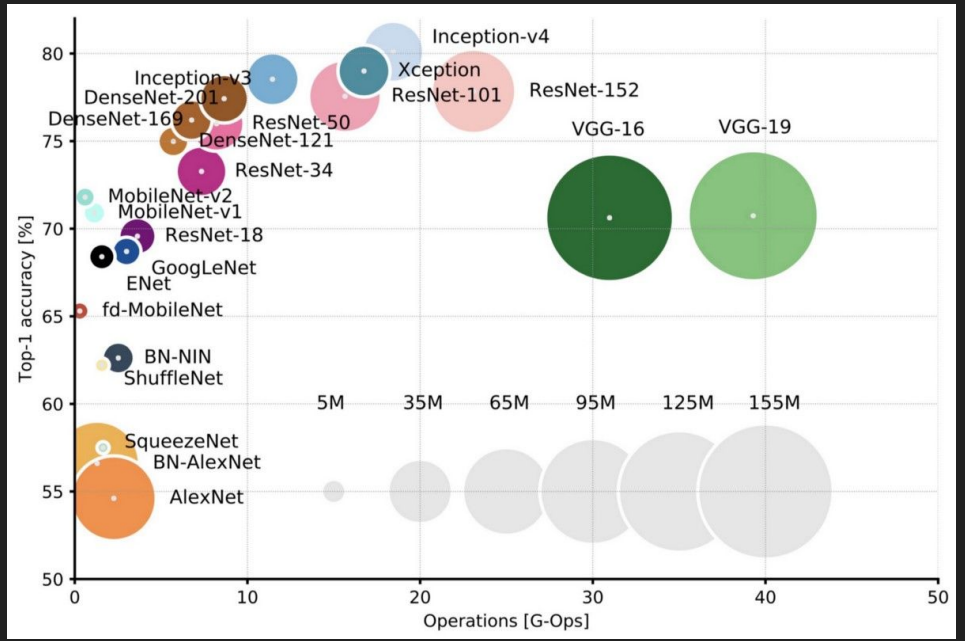
Visual Network

- Initial Plan
  - CNN to creating 2D bounding box around placed pavers
  - Finding position to place next paver
  - Creating 3D bounding box for getting depth of placed paver
  - Send information to autonomous controller



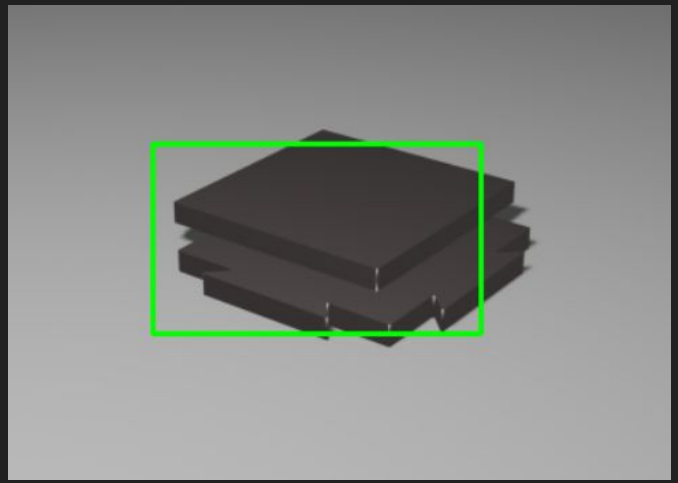
Trained Model

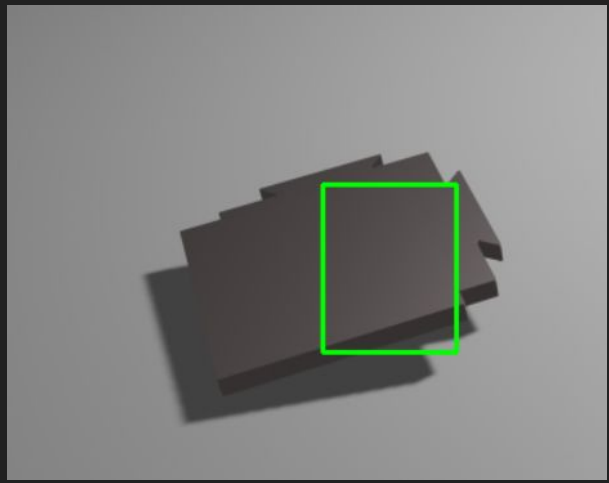
- Technologies
  - Tensorflow and Keras
  - Google Colab
- Training/Testing Images
  - 4,800 images
  - Images split 80% training, 20% testing
- VGG16 Model
  - Optimizer: Adam
  - Loss Function: MSE



Trained Model Results

Epochs: 10 Batch Size: 15 Learning Rate: 0.0001

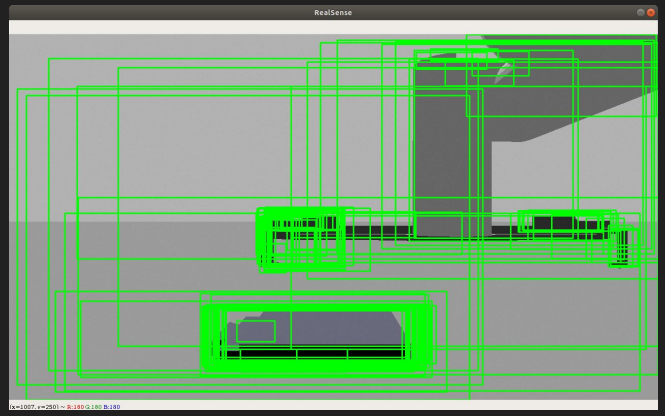


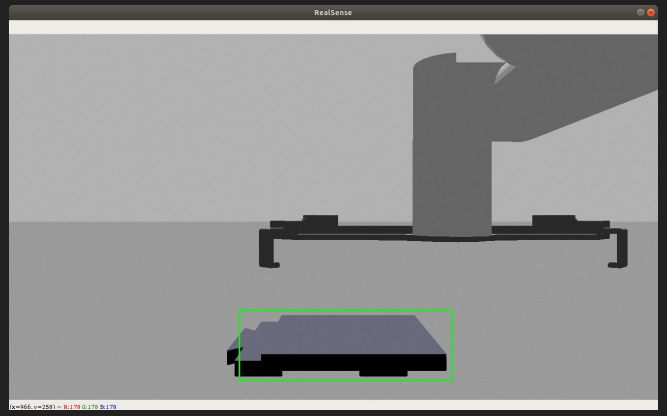


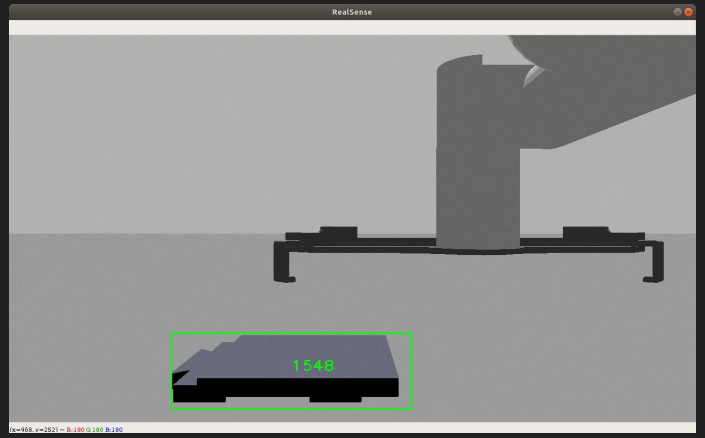
Visual Network

- Final Plan
  - Use pre-trained Faster RCNN Inception model and openCV functions to produce 2D bounding box
  - Calculate center coordinates of the box for most accurate depth
  - Use center coordinates to get depth from depth image.
  - Convert pixel coordinates to physical 3D coordinates
  - Send information to autonomous controller

Final Results

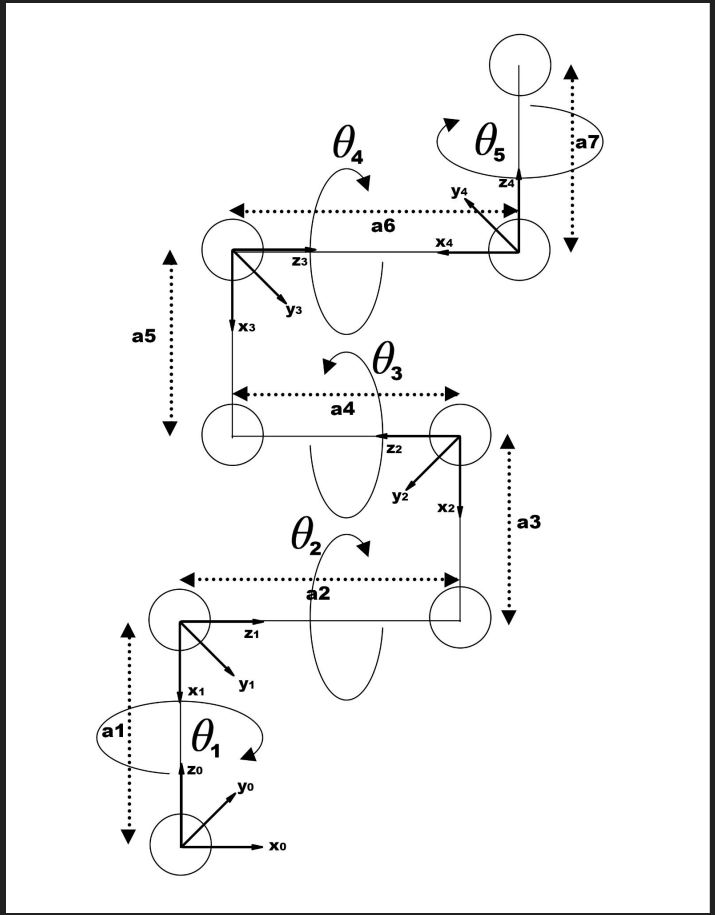






Autonomous Arm Control

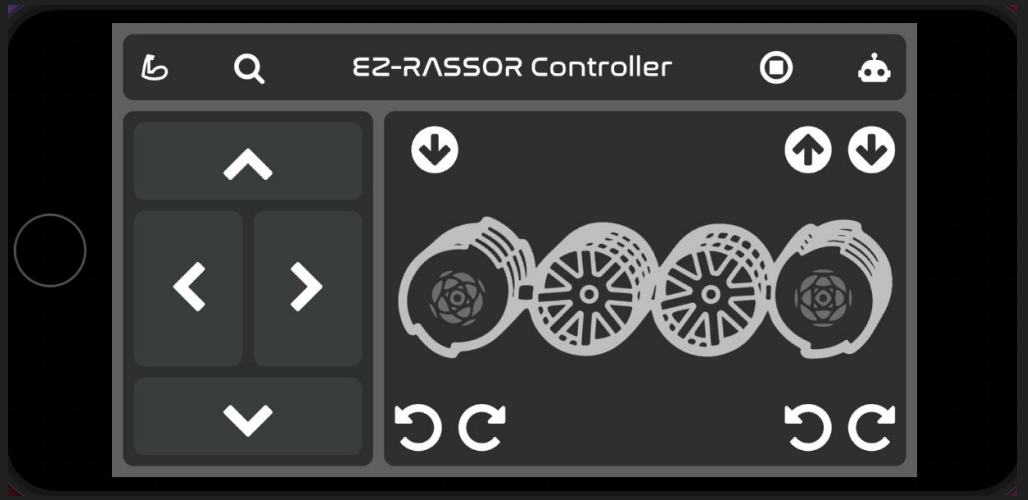
- Moveit Library
  - Ros library for creating custom robotic arm controllers
  - Includes path planning, collision avoidance, end-effector pose goals, joint pose goals
  - Uses inverse kinematics generated from the arm's URDF model to ensure kinematics match the structure of the arm
  - Allows for movement of specific joints, joint goals, pose goals, as well as cartesian paths
  - Ability to save default poses that can be accessed quickly
  - Tools for controlling the speed of the arm by limiting its max velocity/acceleration



Controller

Functionality

- UI for students and users to be able to control the rover and arm
- Users are able to fully control rover in simulation
- Can change which Rover to control within the App using IP address



Controller Milestones

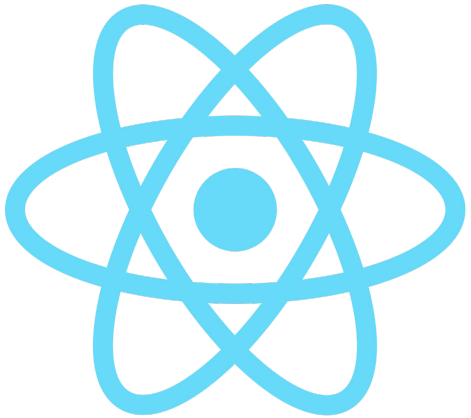
- Added Robotic Arm Screen
- Added navigation between Rover and Arm screens
  - Easier for users
- Changed UI to account for Grabber End rotation
  - More precise movements
- Added autonomous controls for Arm
- Implemented arm commands on the back-end
  - Buttons update rover simulation in real time

Controller Technology

Technologies used

- React native

- Expo/ Expo go



React

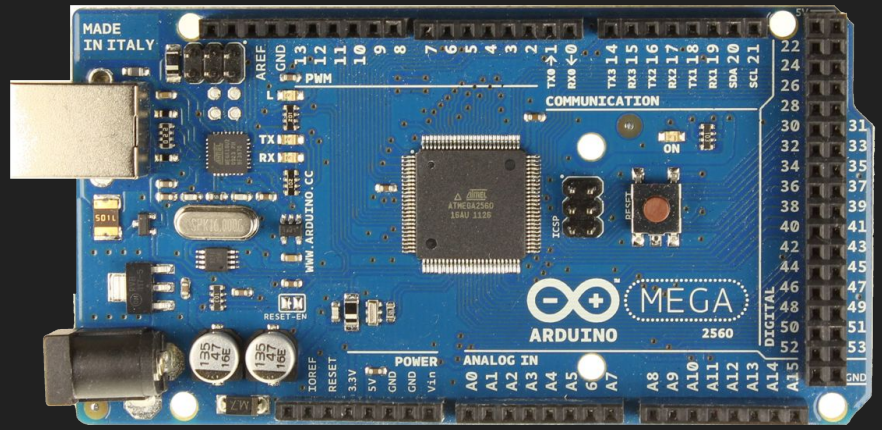
Expo

Physical Rover: Camera

- Intel RealSense D435i
- Stereo vision camera
    - Two cameras allow for detection of all objects and calculating distance
- Used in visual network

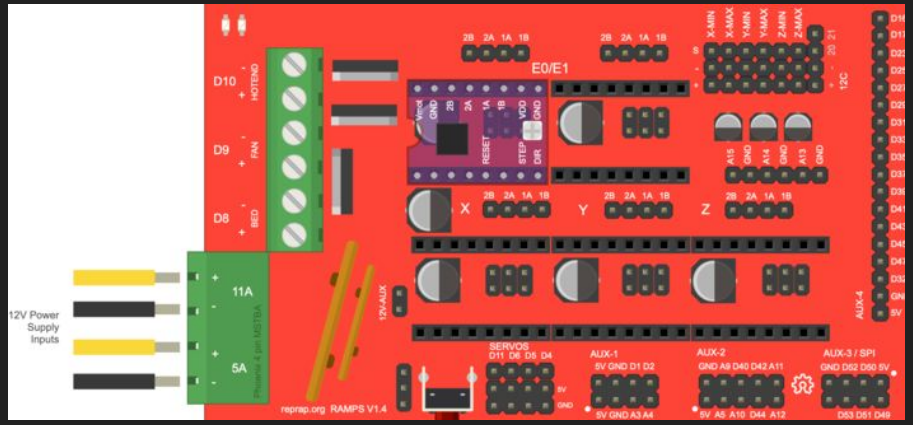
Physical Rover: Arduino

- Arduino Mega 2560 REV3
- Microcontroller board used to operate the rover itself and the arm as well
- Arduino IDE used to make sketches (Arduino programs)
    - Any code made by the user needs only two functions: one to start the sketch and another that is the main program loop
- Arduino sketch will represent state machine for arm, giving it possible motions



Physical Rover: RAMPS 1.4

- RepRap Arduino Mega Polulu Shield 1.4
- Board that serves as the interface between the Arduino Mega and the motors / limit switches
    - Extends functionality by adding 3 thermositers, 5 stepper sockets, heaters & fans, etc.
- Mega 2560 doesn’t have enough pins to operate all of the motors and switches



Physical Rover: Nano

- Nvidia Jetson Nano Developer Kit
- AI computer used to process visual network
- Paver detection and related information will be gathered here
- Communicates with Arduino to place down pavers

Budget: $180

- Software : open source
- Hardware: mostly provided by sponsor
  - Jetson Nano bought by Robert for $180
- 3D printed Physical Arm

Challenges

- Outdated software
  - ROS Melodic
  - Ubuntu 18
  - Deprecated Functions
- Software that isn’t available on all Operating Systems
  - Dual boot!
  - Maybe have an additional Linux Machine
- Difficult problem solving w/ orientation
- Physics Engines

Successes

- Solid start to project:
  - Quick implementation of arm model and manual controls in simulation
- Support/Advice from maintainers and sponsor when facing implementation design challenges
  - A lot of freedom to adjust needs of the arm to allow for the project to succeed
- Numerous libraries found to help facilitate development
  - Moveit, roslaunch, pyrealsense
- Found creative solutions to complex problems
  - Orientation problem solved by using a driving algorithm
  - Physics engine problems solved using dynamic links and bitmask collisions