195

CyberKnights

DESIGN
BUILD
COMPETE
REPEAT

TAKING FRC ROBOTS TO THE NEXT LEVEL WITH ROS

Chris Bonomi

ABOUT ME

- Student on Team 195 from 2015-2018
  - Programming Lead 2016-2018
- Studied Computer Engineering at Florida Institute of Technology
- Software Engineer @ Lockheed Martin
- Current Lead Programming Mentor for Team 195

195

CyberKnights

WHAT IS ROS?

WHAT IS ROS?

- Robot Operating System
- Open source framework for building robotics software
- Modular message based system
- Enables developers to create robust and complex robotic systems, while still maintaining flexibility and scalability

195

CyberKnights

WHY ROS?

WHY ROS?

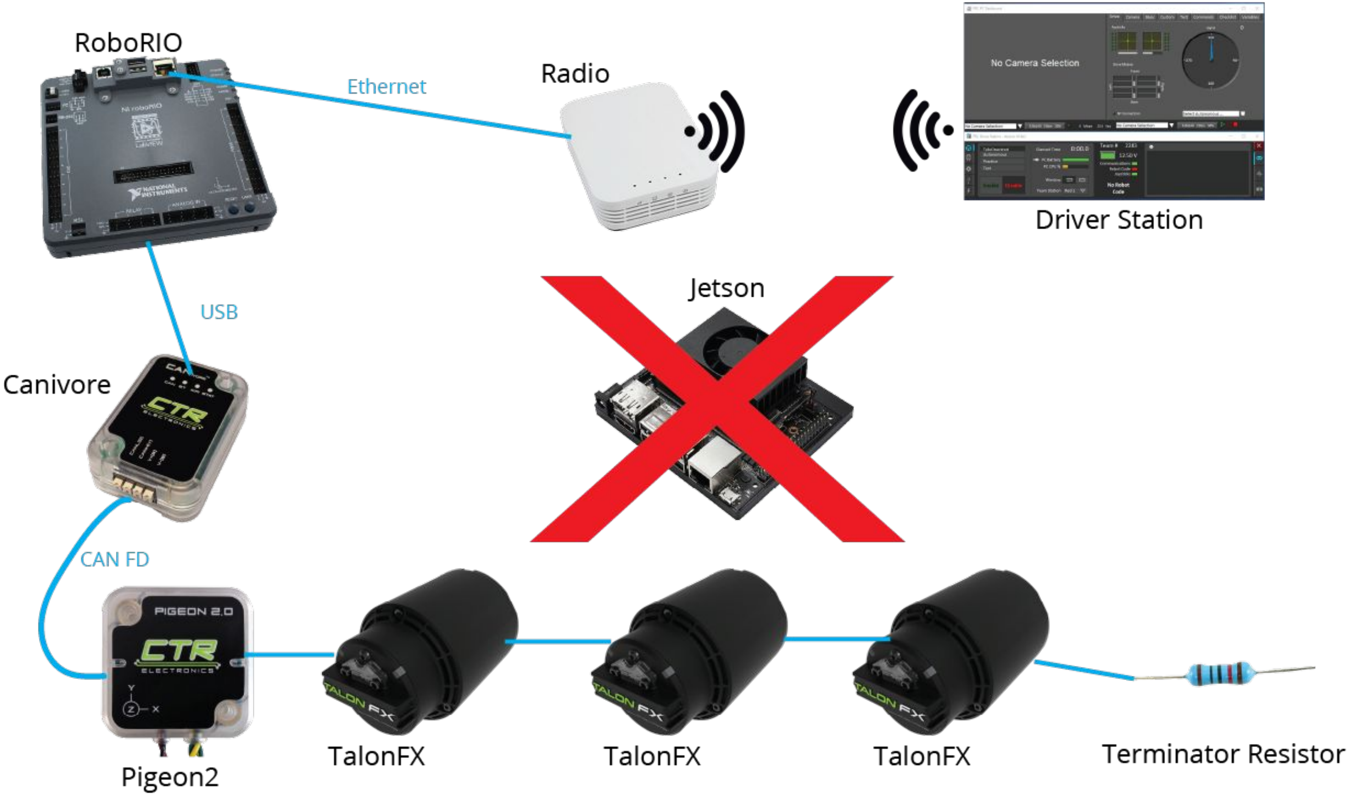
- Industry standard
- Modular code base
- Full C++ & Python support
- Easy data logging and recording
- Open source
- Commercial Friendly
- Large ecosystem of existing packages
- Very active community

195

CyberKnights

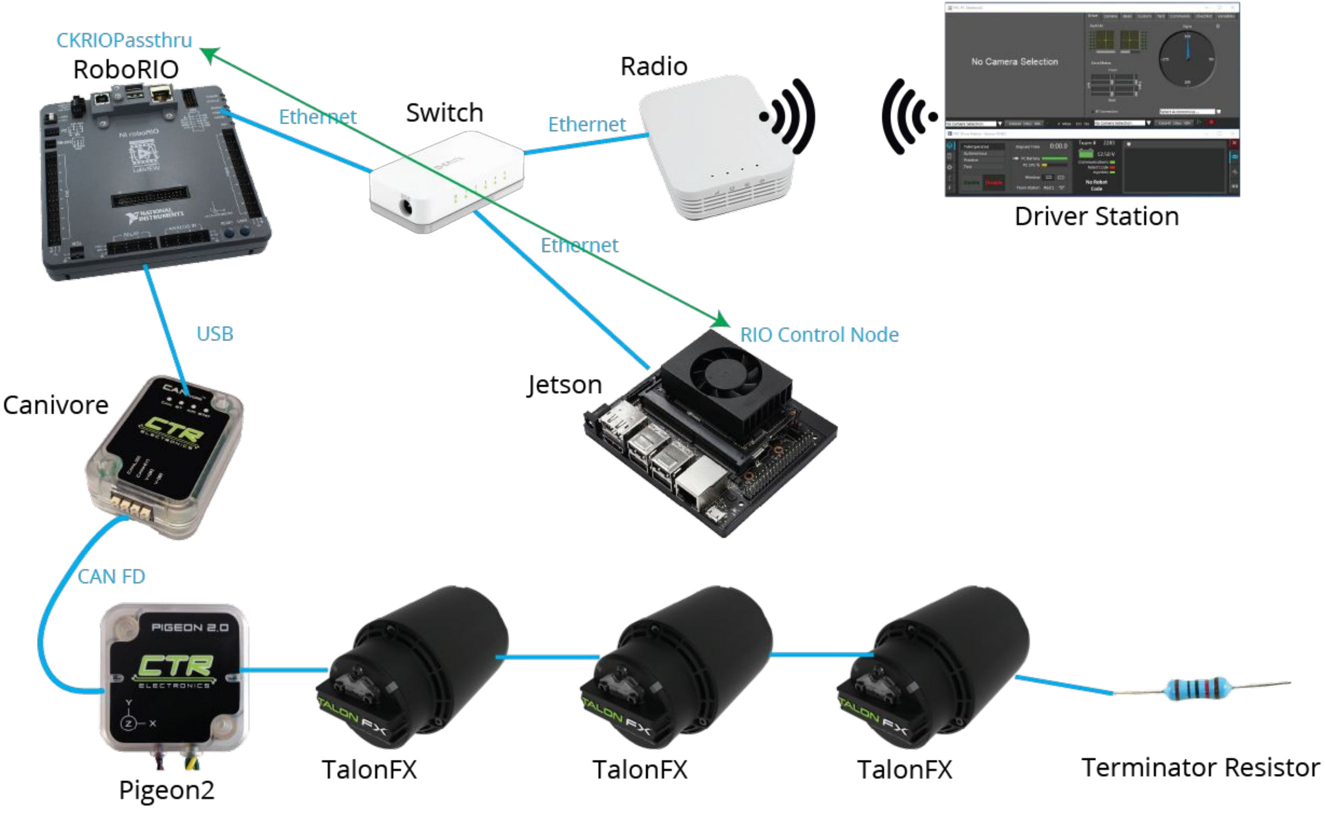
HARDWARE OVERVIEW

Standard Control System



ROS Control System

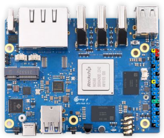
First Gen

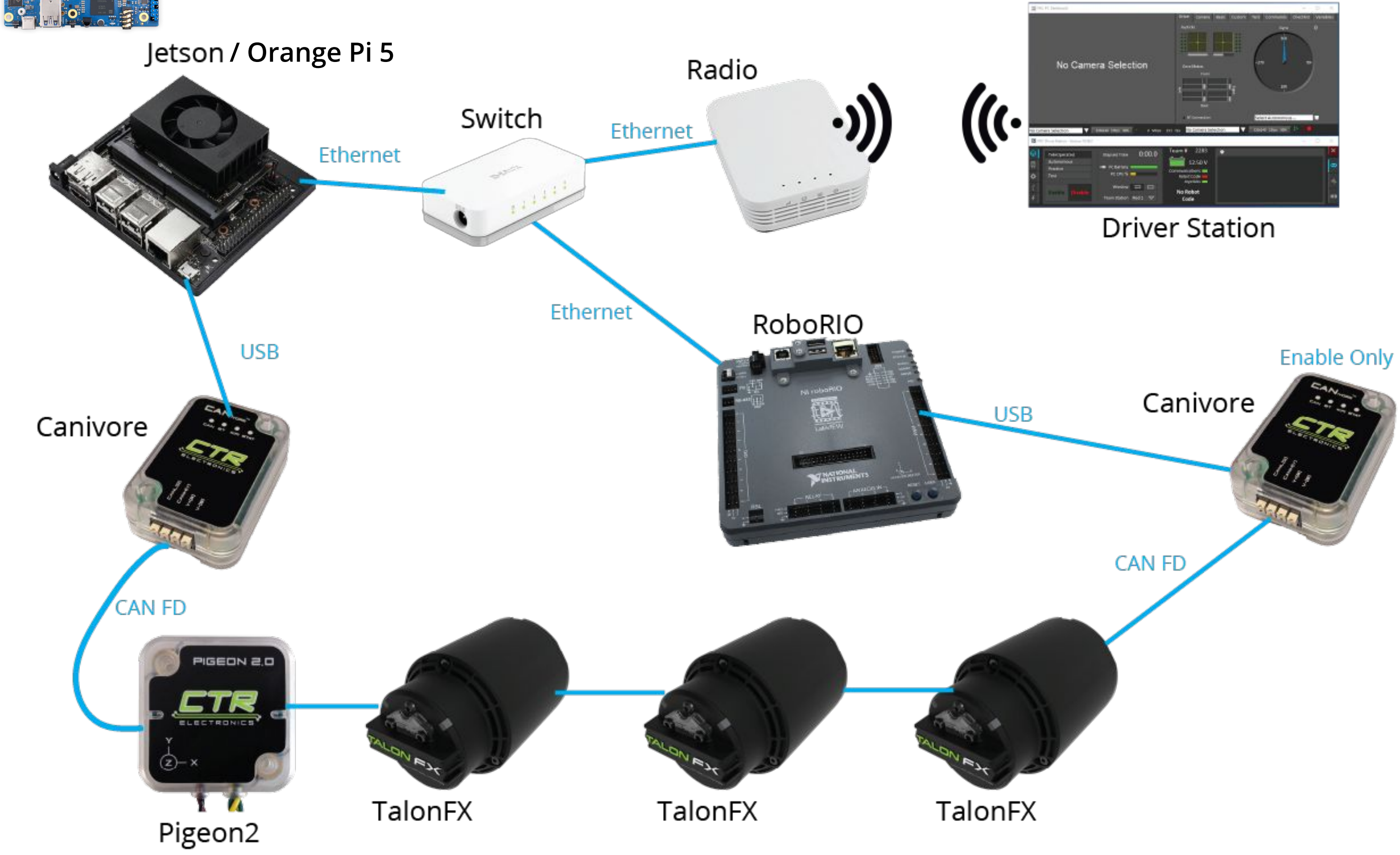


195 CyberKnights

ROS Control System

Second Gen





195
CyberKnights

195

CyberKnights

INTRODUCTION TO ROS2

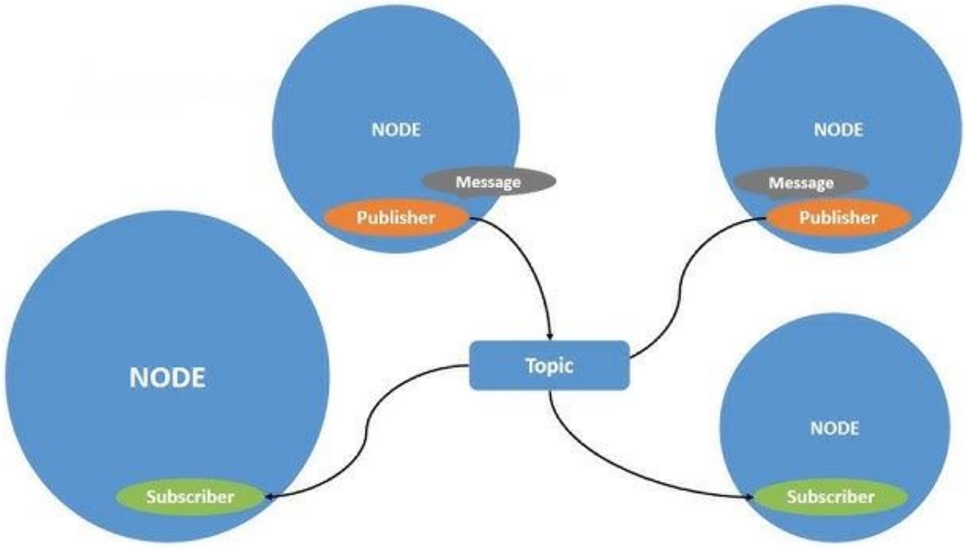
WHAT WE USE

- 2023
  - ROS 1: Noetic Ninjemys
  - Jetson Xavier NX - Not connected directly to CAN
- New for 2024
  - ROS 2: Iron Irwini
  - Jetson Xavier NX - Connected directly to CAN
- Exploring for 2025
  - Orange Pi 5 / 5 Plus
    - Cheaper than the Jetson (~$150 vs. ~$400)
    - More powerful CPU / Non-CUDA Accelerated GPU

1195 CyberKnights

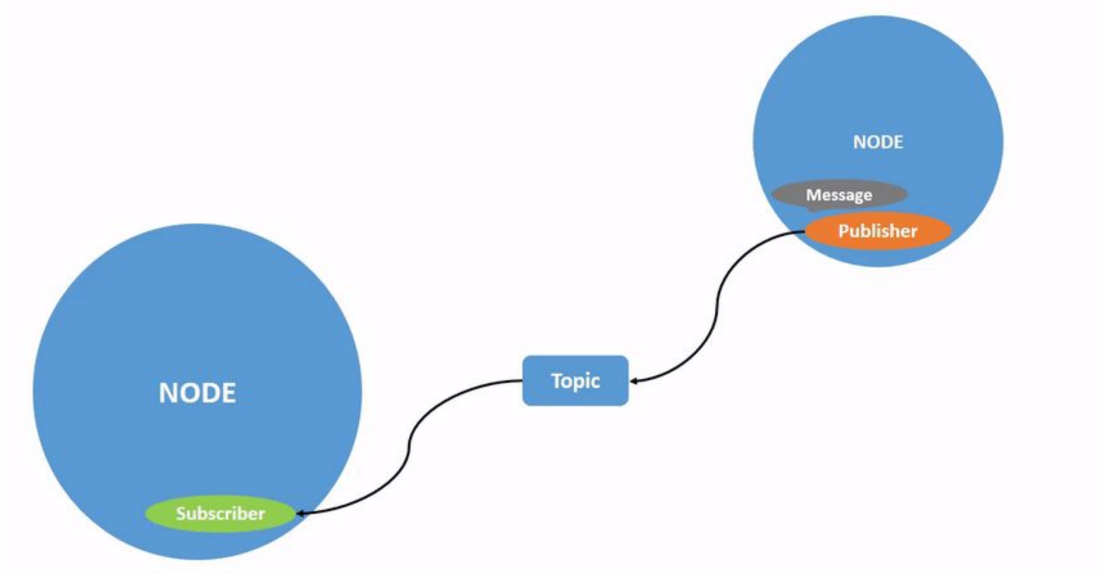
NODE-BASED ARCHITECTURE

- Distributed network of individual processes, performing specific tasks. E.x. Drivebase node, Shooter node, HMI node
- Each node can send a receive data from other nodes via topics, and can be configured with parameters



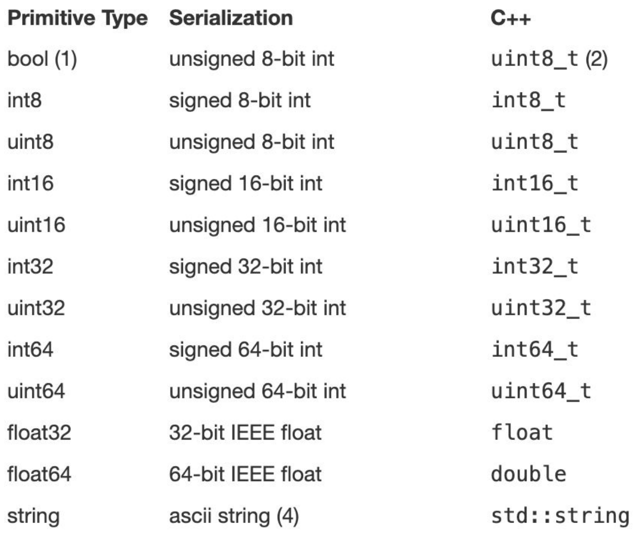
TOPICS

- One of the main ways in which data is exchanged between nodes
- A node may publish data to any number of topics while simultaneously have subscriptions to any number of topics

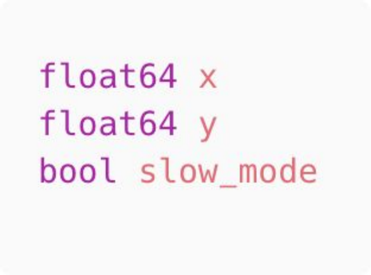


MESSAGES

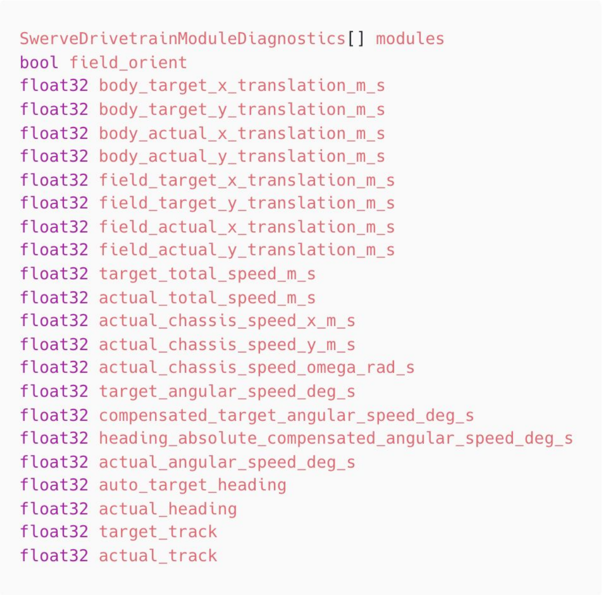
- Data structures that are sent back and forth between nodes across topics, containing information such as sensor data, controller inputs, odometry data
- ROS provides many standard message types, but also supports the creation of completely custom types



Primitive Type	Serialization	C++
bool (1)	unsigned 8-bit int	`uint8_t` (2)
int8	signed 8-bit int	`int8_t`
uint8	unsigned 8-bit int	`uint8_t`
int16	signed 16-bit int	`int16_t`
uint16	unsigned 16-bit int	`uint16_t`
int32	signed 32-bit int	`int32_t`
uint32	unsigned 32-bit int	`uint32_t`
int64	signed 64-bit int	`int64_t`
uint64	unsigned 64-bit int	`uint64_t`
float32	32-bit IEEE float	`float`
float64	64-bit IEEE float	`double`
string	ascii string (4)	`std::string`



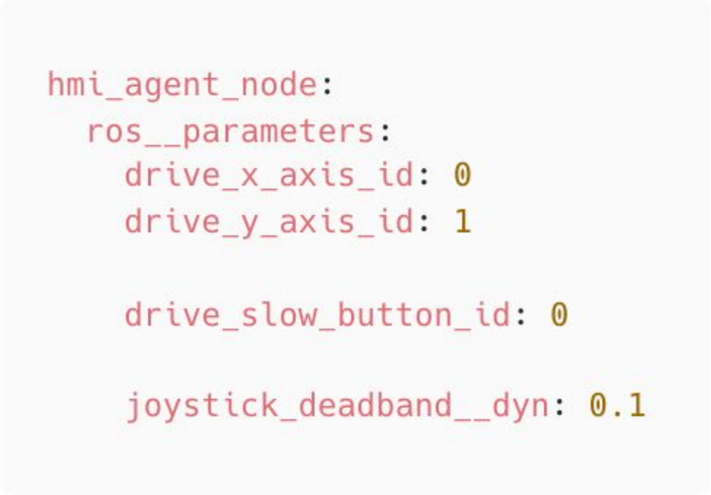
```
float64 x
float64 y
bool slow_mode
```



```
SwerveDrivetrainModuleDiagnostics[] modules
bool field_orient
float32 body_target_x_translation_m_s
float32 body_target_y_translation_m_s
float32 body_actual_x_translation_m_s
float32 body_actual_y_translation_m_s
float32 field_target_x_translation_m_s
float32 field_target_y_translation_m_s
float32 field_actual_x_translation_m_s
float32 field_actual_y_translation_m_s
float32 target_total_speed_m_s
float32 actual_total_speed_m_s
float32 actual_chassis_speed_x_m_s
float32 actual_chassis_speed_y_m_s
float32 actual_chassis_speed_omega_rad_s
float32 target_angular_speed_deg_s
float32 compensated_target_angular_speed_deg_s
float32 heading_absolute_compensated_angular_speed_deg_s
float32 actual_angular_speed_deg_s
float32 auto_target_heading
float32 actual_heading
float32 target_track
float32 actual_track
```

PARAMETERS

- Parameters are configuration values for nodes
- Configured as a YAML file, each node in ROS2 has its own set of parameters
- Supported types are: integers, floats, booleans, strings, and lists
- Parameters may be dynamically modified at runtime, allowing for easy tuning of PID gains, changing of control button layouts, etc.



```yaml
hmi_agent_node:
  ros__parameters:
    drive_x_axis_id: 0
    drive_y_axis_id: 1

    drive_slow_button_id: 0

    joystick_deadband__dyn: 0.1
```

195

CyberKnights

KEY BENEFITS

KEY BENEFITS

- Code Reusability
  - Since each node is fully modular, they can easily be shared between projects or carried over from year to year with very little to no code modification, only parameters
- Fault Tolerance
  - A code failure in a single node will cause only that node to crash, and nodes can be configured to respawn after a crash, so even in the event of a fault the rest of your code can continue to function
- Easy Live Debugging/Recording
  - Programs like PlotJuggler and Foxglove can monitor all topics in real time and plot numeric values
  - Foxglove is even capable of rendering 3D scenes and more (Demo shortly)
  - Topics can also be recorded to a standard MCAP format, for future review to identify errors i.e. failure on the field

195 CyberKnights

DEVELOPMENT ENVIRONMENT

DEVELOPMENT ENVIRONMENT

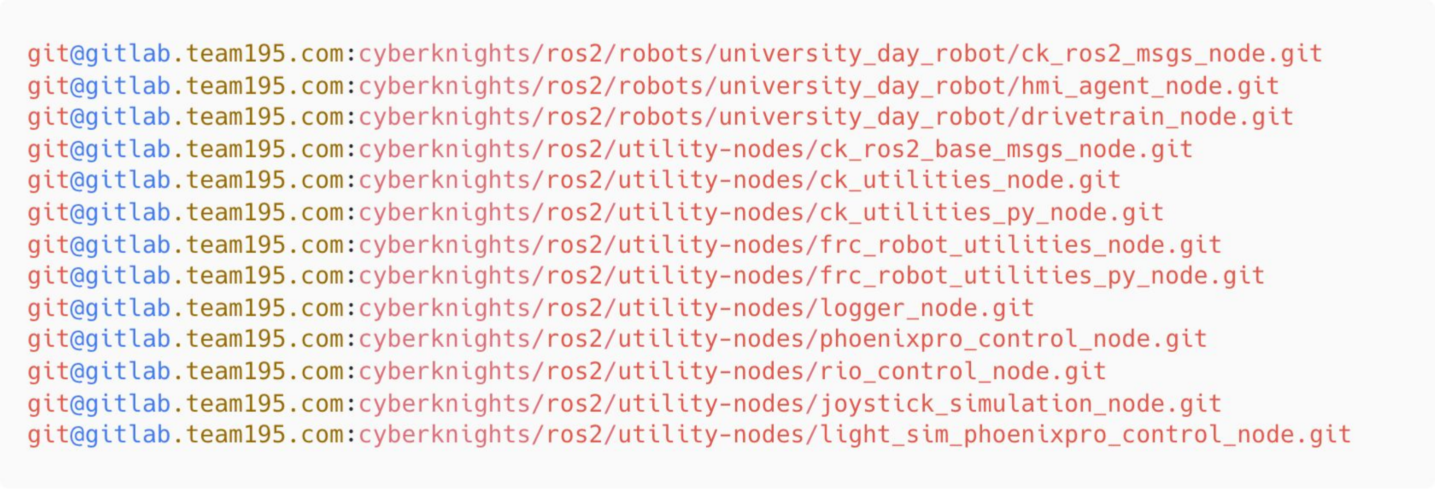
- Ubuntu Virtual Machine
  - 20.04/22.04 to resemble the operating system running on our Jetson
- Docker Container
  - All code is developed and built inside of a docker container, allowing greater control and consistency over our environment from computer to computer
  - The same container is present on the Jetson, and is where ROS runs
- Code Editing
  - All editing is done though Visual Studio Code with the following extensions
    - Dev Containers (Microsoft)
    - C++ Intellisense (Microsoft)
    - Python (Microsoft)

PROJECT LAYOUT

- Flat project structure
- Every robot project starts with 2 sub-projects
  - ros2_dev
    - Contains scripts needed for starting the container and developing/building the robot project
  - A robot sub-project E.x. university_day_robot, 2023_robot
    - Must end in “_robot”
    - Contains launch files, parameter files, and a special file called ros_projects.txt
      - This file defines all other nodes that this robot project requires to run i.e. rio_control_node
- All other node sub-projects are cloned automatically by the mkrobot script

MKROBOT SCRIPT

- One script to handle nearly all project actions
  - Clone all nodes listed in the robots ros_projects.txt file (See example)
  - Update all nodes from git
  - Create new nodes
  - Build/clean project
  - Launch project locally
  - Deploy project to robot



```
git@gitlab.team195.com:cyberknights/ros2/robots/university_day_robot/ck_ros2_msgs_node.git
git@gitlab.team195.com:cyberknights/ros2/robots/university_day_robot/hmi_agent_node.git
git@gitlab.team195.com:cyberknights/ros2/robots/university_day_robot/drivetrain_node.git
git@gitlab.team195.com:cyberknights/ros2/utility-nodes/ck_ros2_base_msgs_node.git
git@gitlab.team195.com:cyberknights/ros2/utility-nodes/ck_utilities_node.git
git@gitlab.team195.com:cyberknights/ros2/utility-nodes/ck_utilities_py_node.git
git@gitlab.team195.com:cyberknights/ros2/utility-nodes/frc_robot_utilities_node.git
git@gitlab.team195.com:cyberknights/ros2/utility-nodes/frc_robot_utilities_py_node.git
git@gitlab.team195.com:cyberknights/ros2/utility-nodes/logger_node.git
git@gitlab.team195.com:cyberknights/ros2/utility-nodes/phoenixpro_control_node.git
git@gitlab.team195.com:cyberknights/ros2/utility-nodes/rio_control_node.git
git@gitlab.team195.com:cyberknights/ros2/utility-nodes/joystick_simulation_node.git
git@gitlab.team195.com:cyberknights/ros2/utility-nodes/light_sim_phoenixpro_control_node.git
```

COMMON PROJECTS

- ckriopassthru
  - This is the project that is deployed to the RoboRIO
  - Sends and receives control signals to the Jetson
  - Serializes data with Protobuf and transmits the data with ZMQ
- rio_control_node
  - This node is responsible for communicating with the rio passthru, acting as the bridge for all other ROS nodes to send/receive commands from the RIO
- phoenix_pro_control_node
  - If using a Canivore with the Jetson, this node sends commands directly to the CTRE CAN devices, without needing to go through rio_control_node and ckriopassthru
- ck_ros2_base_msgs_node
  - This common node contains standard custom message types that are used throughout the entire robot project
- Utility Nodes
  - Common utilities implemented in both C++ and Python to be reused between nodes

195

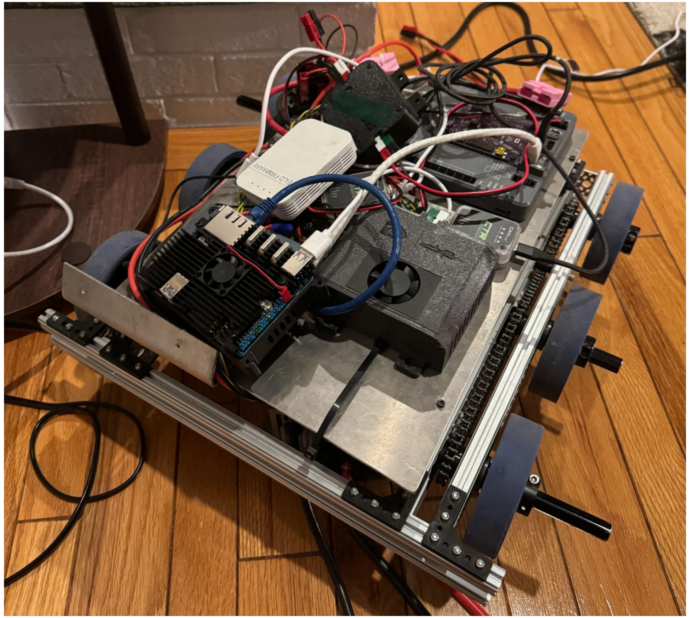
CyberKnights

OUR LITTLE GUY

FLATTY 2!!

- Basic Differential Drive
  - 2 Falcon 500 motors
- Jetson Xavier NX Coprocessor
  - Directly connected to the robot CAN network with a USB Canivore
- VERY Messy Wires

*artists rendition



195

CyberKnights

LIVE CODING!

DELIVERABLES

- drivetrain_node (C++)
  - Subscribe to HMI signals topic to get driver control data
  - Calculate arcade drive outputs for left and right motors and apply them
  - Publish a diagnostic message to be plotted in Foxglove

- hmi_agent_node (Python)
  - Subscribe to Joystick status topic to get current joystick inputs
  - Read the values of the desired axes and buttons
  - Publish the desired control inputs on the HMI topic

195

CyberKnights

QUESTIONS?

Email:

programming@team195.com

GitLab: