2019 Code Documentation

**** This document is for detailing how the code is put together. Specific details can be found in the comments within the code. This documented should be updated anytime major changes occur with the code!***

:: Header Files ::

Robot.h

- Contains the Robot Class declaration, which inherits the **Timed Robot** Class

Excelsior_Classes.h

- Contains the Custom Class declarations, which include:
  - **Excelsior_Omni_Drive** – All actions associated with drivetrain
  - **Excelsior_Payload_Lift** – All actions associated with the payload lifting mechanism
  - **Excelsior_End_Effector** – All actions associated with the end effector mechanisms
- Contains **#definitions** for Joystick button and axis mapping for a Gamepad Controller
- Contains the **enum** for the Payload Lift positions

:: C++ Files ::

Robot.cpp (the main code file)

- Include Section:
  - Header files for local code (Robot.h, Excelsior_Classes.h)
  - Header files for external code (frc/WPILib.h)
  - Code files for C++ native classes (iostream)
- **Tuning Variables Section**:
  - Joystick channels, included in this section in case we want to swap the controllers for testing
  - Enabling bits for testing mechanisms (Drivetrain, Lift, and End Effector)
  - Enabling bit for printing encoder values from any active TalonSRX devices
  - Deadband trigger is used to reject small movements of the Gamepad trigger axis'
- **Control Logic Switchboard Sections**:
  - Omni Drive:
    - Set the control scheme for driving the wheels and switching to manual
  - Payload Lift:
    - Set the control scheme for Lift positions, position stepping, and switching to manual
    - Set the control scheme for zeroing the encoder (which should only happen once ideally)
  - End Effector:
    - Definition for getting raw Gamepad trigger values which are used in later schemes
    - Set the control scheme for Cargo Roller driving and switching to manual
    - Set the control scheme for the Hatch Flower and Camera Tilt servos
- Declarations Section:
  - Declare motor and joystick class objects, as well as any global variables
- **Teleop Periodic Section**:

    - Omni Drive:
        - Pass the drivetrain control scheme to the **Action** method
    - Payload Lift:
        - First check if the operator is activating the zeroing action... call the **Zero_Encoder** method
        - Next check if the operator is activating any of the lift positions… call the **Action** method
        - Next check if the operator is activating either of the step actions… call the **Step** method
            - Notice the STEP actions use a variable (pressedLastFrame_auto), which ensures the position is stepped only once until the operator releases the action button
        - Next check if the operator is activating any manual actions… call the **Manual** method
            - Notice the MANUAL action also uses a variable (pressedLastFrame_manual), which ensures that the lift arm is stopped after the operator releases the action button
        - ELSE… reset the pressedLastFrame_auto variable (operator has released the button)
        - Check if the operator has released the manual lift button… call the **Manual** method with **0 speed**
        - Check if encoder printing is enabled… call the **Print Encoder** values method
    - End Effector:
        - First check if the operator is activating the manual cargo roller action… call the **Manual** method
            - Notice this uses another (pressedLastFrame_cargoRollers), but a little different
        - Next check if the operator has released the manual cargo roller action button
        - Next check if the operator is activating normal cargo actions… call the **Roller Action** method
        - Next check if the operator is activating the hatch servo… call the **Hatch Action** method
        - Check if the operator is activating the camera servo… call the **Camera Action** method
        - Check if encoder printing is enabled… call the **Print Encoder** values method
- Robot Init Section:
    - Call the **Configure** method of each mechanism class
- Misc Method Section:
    - These are the Robot class methods we are not using, but are available to us if we need, and are required to be included in the code to satisfy the Timed Robot class inheritance
- Start Robot Section:
    - This is the **MAIN function**, which contains the actual work of running the Robot Class

Omni_Drive.cpp

- Include Section:
    - Header files for local code (Excelsior_Classes.h)
    - Header files for external code (frc/WPILib.h, frc/Talon.h, ctre/Phoenix.h)
    - Code files for C++ native classes (iostream, Math.h)
- **Tuning Variables Section**:
    - Omni Drive speed variable, which needs to be tuned for a desired robot speed
    - **Configuration**!
        - Enabling bit to write the configuration to the TalonSRX devices, which should only happen AS NEEDED when changes are required for the motion profile
        - The Motion Profile consists of PID variables P (proportional), D (derivative), F (feed forward), Peak output forward and reverse, and the ramp time
    - Deadband value is to reject small movements of the Gamepad wheels during operation
- Definitions Section:
    - CAN device numbers for motor controllers

    - Convert to RPS is a scalar and was originally computed using a ToughBox, see comment
- Declarations Section:
    - Declare motors and any global variables (none currently)
- Configure Section:
    - Set the Leader-Follower logic, motor orientation (inverted), and Configuration if enabled
- **Action Section**:
    - Compute the direction vector and scalar, and rotation correction, as new output speeds
    - Write the output speeds normally as **Velocity** control, otherwise manual as **PercentOuput**
- Helper Functions Section:
    - Currently just the encoder printing method

Payload_Lift.cpp

- Include Section:
    - Same as Omni_Drive.cpp, except using a C++ native class (map) and not using (Math.h)
- **Tuning Variables Section**:
    - **Configuration** for TalonSRX's same as Omni_Drive.cpp, see above
- Definitions Section:
    - CAN device numbers for motor controllers
    - The position scalar variable is for unique encoder values within one rotation of output shaft
- Declarations Section:
    - The **MAP** for the lift positions works just like a dictionary, and in this case the KEY is the enum position, and the VALUE is the actual encoder position
    - Declare motors and any global variables (just the targetPayloadHeight right now)
- Configure Section:
    - Set the Leader-Follower logic, motor orientation (inverted), and Configuration if enabled
- **Action Section**:
    - Set the new lift position using **Position** control, and save that enum position as an integer which is used for stepping
- Manual Section:
    - Use **PercentOutput** control to drive the lift
- Step Section:
    - Move the lift up or down one position as specified by the enum ordering, making sure we don't exceed the upper (max height) and lower (ground level) limits
- Zero Encoder Section:
    - Zero the encoder position, which should happen when the arm is at ground position
- Helper Functions Section:
    - Currently just the encoder printing method

End_Effector.cpp

- Include Section:
    - Same as Omni_Drive.cpp, but also includes an external class (frc/PWM.h) and doesn't use (Math.h)
- **Tuning Variables Section**:
    - Limit switch enabling bit, which can be used to turn on/off cargo catch limit switch protection
    - Roller speed RPS needs to be tuned for the desired lift speed
    - **Configuration** for TalonSRX's same as Omni_Drive.cpp, see above

    - Hatch Flower servo position MIN/MAX needs to be tuned for the desirable OUT and IN servo positions
    - Camera Tilt servo position MIN/MAX needs to be tuned for desirable UP and DOWN servo positions
- Definitions Section:
    - CAN device numbers for motor controllers
    - PWM channels for Hatch servo and Camera servo
    - DigitalIO channel for the limit switch
    - Convert to RPS is a scalar and was originally computed using a ToughBox, see comment
- Declarations Section:
    - Declare motors, PWM devices, DigitalIO devices, and any global variables (none currently)
- Configure Section:
    - Set the configuration for the encoder, as well as the orientation (polarity)
    - Set the Leader-Follower logic, motor orientation (inverted), and Configuration if enabled
- **Roller Action Section:**
    - Use **Velocity** control to set output speed
- Manual Section:
    - Use **PercentOutput** control to set output speed
- **Hatch Action Section**:
    - Set servo IN or OUT
- **Camera Tilt Section**:
    - Move the servo position to the UP or DOWN position
- Helper Functions Section:
    - Currently just the encoder printing method