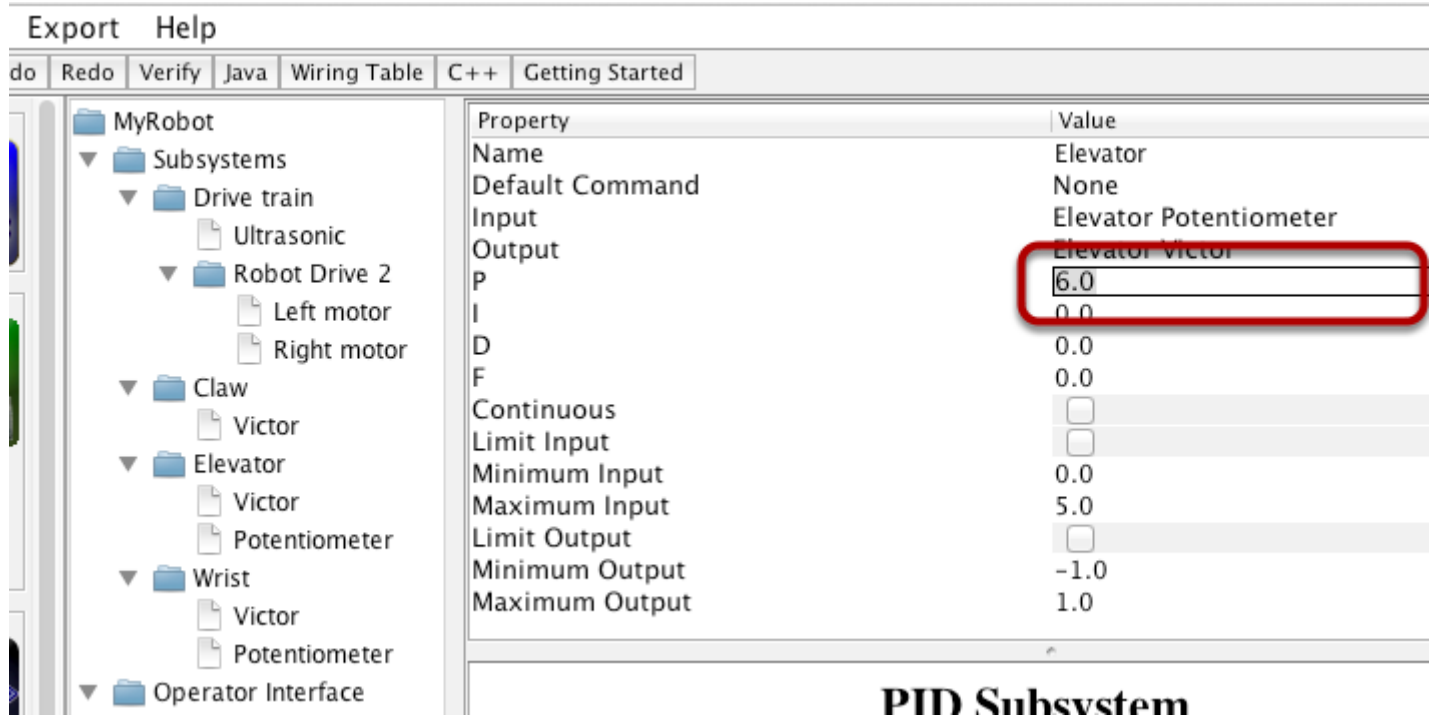


Writing the code for a PIDSubsystem in C++

Writing the code for a PIDSubsystem in C++

Setting the PID constants



Make sure the Elevator PID subsystem has been created in the RobotBuilder. In the case of our elevator we use a proportional constant of 6.0 and 0 for the I and D terms. Once it's all set, generate C++ code for the project using the Export menu or the C++ toolbar menu.

Writing the code for a PIDSubsystem in C++

Add constants for the Elevator preset positions



```
#ifndef ELEVATOR_H
#define ELEVATOR_H

#include "Commands/PIDSubsystem.h"
#include "WPILib.h"

class Elevator: public PIDSubsystem {
public:
    static const double BOTTOM = 4.6;
    static const double STOW = 1.65;
    static const double TABLE_HEIGHT = 1.58;

    // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DECLARATIONS
    AnalogChannel* potentiometer;
    Victor* victor;
    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DECLARATIONS
    Elevator();
    double ReturnPIDInput();
    void UsePIDOutput(double output);
    void InitDefaultCommand();
};

#endif
```

Elevator constants define potentiometer voltages that correspond to fixed positions on the elevator. These values can be determined using the print statements, the LiveWindow or SmartDashboard.

Writing the code for a PIDSubsystem in C++

Initialize the elevator position in the Elevator constructor

```
#include "Elevator.h"
#include "../RobotMap.h"
#include "SmartDashboard/SmartDashboard.h"
// BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=PID
Elevator::Elevator() : PIDSubsystem("Elevator", 1.0, 0.0, 0.0) {
    GetPIDController() -> SetContinuous(false);

    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=PID
    // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DECLARATIONS
    potentiometer = RobotMap::ELEVATOR_POTENTIOMETER;
    victor = RobotMap::ELEVATOR_VICTOR;

    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DECLARATIONS
    SetSetpoint(STOW);
    Enable();
}
```

Set the elevator initial position so when the robot starts up it will move to that position. This will get the robot to a known starting point. Then enable the PIDController that is part of the PIDSubsystem. The elevator won't actually move until the robot itself is enabled because the motor outputs are initially off, but when the robot is enabled, the PID controller will already be running and the elevator will move to the "STOW" starting position.

The autogenerated code to return the PID input values

```
double Elevator::ReturnPIDInput() {
    // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=SOURCE
    return potentiometer->PIDGet();
    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=SOURCE
}
```

If you look at the RobotBuilder generated code for ReturnPIDInput() you can see that there is a //BEGIN and //END comment delimiting the return statement. \ In between //BEGIN and //END comments is where RobotBuilder will rewrite code on subsequent exports. So in general you should not change any code in that block. But, the function returns the value in raw analog units (0-1023). The setpoints are in units of Volts, so this won't work.

Writing the code for a PIDSubsystem in C++

Set the ReturnPIDInput method to return voltage

```
double Elevator::ReturnPIDInput() {  
    return potentiometer->GetAverageVoltage();  
}
```

The function must be changed to return voltage. If we change the code inside the //BEGIN and //END block, it will just be overwritten next time RobotBuilder exports the file. The solution is to remove the //BEGIN and //END comments, then make the change. This will prevent RobotBuilder from changing the code back again later.

That's all that is required to create the Elevator PIDSubsystem in C++. To operate it with commands to actually control the motion see: [Operating a PIDSubsystem from a command in C++](#).