# WPI Simulation Patches

WPILIB 2016-2017

## Table of Contents

## Build Details

1. allwpilib

   For simulation the allwpilib project needs to be built from an Ubuntu 14.04 operating system environment (either native or using a virtual machine)

   The allwpilib build process uses "gradle". Instructions can be found in the README.md file in the top level directory.

   - ./gradlew build -PmakeSim    (creates build/install sub-directory)
   - ./gradlew wpilibc:allcsim -PmakeSim (builds libraries)
   - ./gradlew simulation:zip -PmakeSim (packages header files into a single directory)

   Following the above operations most of the files needed for simulation will be present in allwpilib/build/install/simulation but to make a complete tar file for export the gz_msgs lib and header files should also be built and copied into this directory

   - ./gradlew wpilibc:gz_msgs -PmakeSim
   - cp -R ~/allwpilib/build/simulation/gz_msgs/generated/simulation/gz_msgs ~/allwpilib/build/install/simulation/include
   - cp ~/allwpilib/build/simulation/gz_msgs/libgz_msgs.so ~/allwpilib/build/install/simulation/lib

## Installation

1. For convenience a tar file (wpilib.simulation.tar) that includes linux libraries and header files simulation has been placed in ~/Robotics/MentorRepository/patches
2. To install in Ubuntu, untar wpilib.simulation.tar file into  ~/wpilib (e.g. tar -xf wpilib.simulation.tar -C ~/wpilib)

## Patches

### Simulation Problems and workarounds

1. wpilibc/sim/include/Counter.h

- Problem: causes build failure
- Patch:

  undefined symbol “Mode”:  in  Counter.h (Mode mode = kTwoPulse not defined)

```
@@ -30,7 +30,7 @@ class Counter : public SensorBase,
                public CounterBase,
                public LiveWindowSendable {
  public:
-  explicit Counter(Mode mode = kTwoPulse);
+  //explicit Counter(Mode mode = kTwoPulse);
```

2. wpilibc/simulation/src/Notifier.cpp

- Problem: causes multiple PIDControllers to fail to report correct updates
- Patch:

```
@@ -151,6 +151,9 @@ void Notifier::InsertInQueue(bool reschedule) {
     if ((*i)->m_expirationTime > m_expirationTime) {
       timerQueue.insert(i, this);
       m_queued = true;
+                    // BUG 1: wpi version doesn't break here so it keeps inserting in front of all elements
+                    // with expiration times > current element
+                    break;
     }
   }

@@ -158,7 +161,8 @@ void Notifier::InsertInQueue(bool reschedule) {
    * element was greater than the new entry.
    */
   if (!m_queued) {
-    timerQueue.push_front(this);
+    // BUG 2: wpi version uses "push_front" which is wrong since it adds the longest time to the front
+              timerQueue.push_back(this);
```

3. wpilibc/simulation/src/PIDController.cpp

- Problem: Only a single simulated PIDController in a robot design works correctly
- Patch:

```
@@ -83,6 +83,8 @@ PIDController::PIDController(double Kp, double Ki, double Kd, double Kf,
   m_controlLoop = std::make_unique<Notifier>(&PIDController::Calculate, this);
   m_controlLoop->StartPeriodic(m_period);
+  m_setpointTimer.Start();
```

```
+
   static int instances = 0;
   instances++;
@@ -172,6 +174,17 @@ void PIDController::Calculate() {
     }
     pidOutput->PIDWrite(result);
+
+               // BUG: 2016 WPI code does not do this
+           // Update the buffer.
+
+           m_buf.push(m_error);
+           m_bufTotal += m_error;
+           // Remove old elements when buffer is full.
+           if (m_buf.size() > m_bufLength) {
+             m_bufTotal -= m_buf.front();
+             m_buf.pop();
+           }
   }
 }

@@ -193,7 +206,7 @@ void PIDController::Calculate() {
 double PIDController::CalculateFeedForward() {
   if (m_pidInput->GetPIDSourceType() == PIDSourceType::kRate) {
     return m_F * GetSetpoint();
-  } else {
+  } else if( m_F>0) {
     double temp = m_F * GetDeltaSetpoint();
     m_prevSetpoint = m_setpoint;
     m_setpointTimer.Reset();
+} else
+    return 0;
@@ -507,8 +520,11 @@ void PIDController::SetToleranceBuffer(int bufLength) {
 bool PIDController::OnTarget() const {
   std::lock_guard<priority_recursive_mutex> sync(m_mutex);
-  if (m_buf.size() == 0) return false;
-  double error = GetError();
+       double error;
+       if (m_buf.size() == 0)
+               error = GetError();
+       else
+               error = GetAvgError();
```

4. wpilibc/sim/src/simulation/SimContinuousOutput.cpp

- Problem: GetSpeed() always returns “0”
- Patch:

```
@@ -16,7 +16,8 @@ SimContinuousOutput::SimContinuousOutput(std::string topic) {
  std::cout << "Initialized ~/simulator/" + topic << std::endl;
 }
-void SimContinuousOutput::Set(double speed) {
+void SimContinuousOutput::Set(double _speed) {
+  speed=_speed;
```

5. wpilibc/sim/src/simulation/SimEncoder.cpp

- Problem: Reset doesn't immediately clear encoder position value
  - Instead, it waits for a reset message to come back from the encoder plugin
    - If encoder position is read back shortly after reset is called it is not zero
  - leads to spikes in position plots
- Patch:
  - add  position=0 to reset call

    ```
    index 141a371..f1c3ddc 100644
    --- a/wpilibc/sim/src/simulation/SimEncoder.cpp
    +++ b/wpilibc/sim/src/simulation/SimEncoder.cpp
    @@ -30,7 +30,7 @@ SimEncoder::SimEncoder(std::string topic) {
    -void SimEncoder::Reset() { sendCommand("reset"); }
    +void SimEncoder::Reset() { sendCommand("reset"); position=0; }
    ```
- build/install
  - cd ~/Robotics/allwpilib-2017
  - edit  wpilibc/sim/src/simulation/SimEncoder.cpp
    - line 33: void SimEncoder::Reset() { sendCommand("reset"); position=0;}
  - gradlew wpilibc:build -PmakeSim
  - cp ./build/install/simulation/lib/libwpilibcSim.so /usr/local/wpi/2017/sim/lib

## Bug Reports submitted to WPI

Title: PIDController class doesn't work in linux_simulation build (c++)

Description

Found a number of problems when trying to create PIDControllers in robot code used for gazebo simulation
1) PIDController.::CalculateFeedForward returns NaN (which makes Calculate useless even if m_F=0)
 - problem is that GetDeltaSetpoint has divide by zero since m_setpointTimer is never started
 - fix was to add "m_setpointTimer.Start();" in Initialize function
2) PIDController::Ontarget always returns false
 - Problem is that OnTarget has "if (m_buf.size() == 0) return false;" which always passes since m_buff is never filled
 - fix was to add "m_buf.push(m_error);" etc. at bottom of Calculate (i.e. same as in Athena code)
 - also added " m_buf = std::queue<double>();" to SetSetpoint
3) Notifier::InsertInQueue has two problems
 - need to add "break;" after "timerQueue.insert(i, this);" test first passes otherwise Notifier is inserted multiple times
 - in same function, below "if (!m_queued)" current code has "timerQueue.push_front(this);" which causes the notifier with the LONGEST time to be pushed to the front (which is supposed to contain notifier with the SHORTEST time)
- this prevents more than a single Notifier (or active PIDController) from being serviced by the timer thread
- fix was to change "timerQueue.push_front(this);" to "timerQueue.push_back(this);"

Title: Can't read speed or voltage back from simulated motors

Description

Calling Talon::Get() or Talon::GetSpeed() always returns 0 in simulation mode

It looks the problem is in SimContinuousOutput.::Set which publishes a gazebo message but never copies the result to it's private speed variable (which is returned by Get)

The fix was add the following change to wpilibc/simulation/src/simulation/SimContinuousOutput.cpp

```
void SimContinuousOutput::Set(float _speed) {
	speed=_speed;
```