Simple Control Loop Example

Overview

This example will show you how to process kobuki's pose data and send wheel commands to robot.

Source Code

Here is full source code of simple control loop example. You can find it from **src/test/simple_loop.cpp**

```
/*****************************************************************************
 Includes
 ****************************************************************************/

#include <csignal>
#include <ecl/time.hpp>
#include <ecl/sigslots.hpp>
#include <ecl/geometry/pose2d.hpp>
#include <ecl/linear_algebra.hpp>
#include "kobuki_driver/kobuki.hpp"

/*****************************************************************************
 Classes
*****************************************************************************/

class KobukiManager {
public:
  KobukiManager() :
    dx(0.0), dth(0.0),
    slot_stream_data(&KobukiManager::processStreamData, *this)
  {
    kobuki::Parameters parameters;
    parameters.sigslots_namespace = "/kobuki";
    parameters.device_port = "/dev/kobuki";
    parameters.enable_acceleration_limiter = false;
    kobuki.init(parameters);
    kobuki.enable();
    slot_stream_data.connect("/kobuki/stream_data");
  }

  ~KobukiManager() {
    kobuki.setBaseControl(0,0); // linear_velocity, angular_velocity in (m/s), (rad/s)
    kobuki.disable();
  }

  void processStreamData() {
    ecl::Pose2D<double> pose_update;
    ecl::linear_algebra::Vector3d pose_update_rates;
    kobuki.updateOdometry(pose_update, pose_update_rates);
    pose *= pose_update;
    dx += pose_update.x();
    dth += pose_update.heading();
    processMotion();
  }
  // Generate square motion
  void processMotion() {
    if (dx >= 1.0 && dth >= ecl::pi/2.0) { dx=0.0; dth=0.0; kobuki.setBaseControl(0.0, 0.0);
return; }
    else if (dx >= 1.0) { kobuki.setBaseControl(0.0, 3.3); return; }
    else { kobuki.setBaseControl(0.3, 0.0); return; }
  }

  ecl::Pose2D<double> getPose() {
    return pose;
  }

private:
  double dx, dth;
  ecl::Pose2D<double> pose;
  kobuki::Kobuki kobuki;
  ecl::Slot<> slot_stream_data;
};

/*****************************************************************************
 Signal Handler
*****************************************************************************/

bool shutdown_req = false;
void signalHandler(int signum) {
  shutdown_req = true;
}

/*****************************************************************************
 Main
*****************************************************************************/

int main(int argc, char** argv)
{
  signal(SIGINT, signalHandler);

  std::cout << "Demo : Example of simple control loop." << std::endl;
  KobukiManager kobuki_manager;

  ecl::Sleep sleep(1);
  ecl::Pose2D<double> pose;
  try {
    while (!shutdown_req){
      sleep();
      pose = kobuki_manager.getPose();
      std::cout << "current pose: [" << pose.x() << ", " << pose.y() << ", " <<
pose.heading() << "]" << std::endl;
    }
  } catch ( ecl::StandardException &e ) {
    std::cout << e.what();
  }
  return 0;
}
```

Explanation

This simple control loop program process kobuki's stream data to retrive how far moved from last stream, and accumulate it to get current pose(position and orientation). And control the robot to following the simple squre path where each side is 1.0 meter.

Includes

```
#include <csignal>
#include <ecl/time.hpp>
#include <ecl/sigslots.hpp>
#include <ecl/geometry/pose2d.hpp>
#include <ecl/linear_algebra.hpp>
#include "kobuki_driver/kobuki.hpp"
```

ecl::Sleep from ecl/time.hpp is used to throttle the rate of outputs of current pose printing. Signal handler from csignal also introduced to prevent unexpected behavior, when it quiting.

ecl/geometry/pose2d.hpp and ecl/linear_algebra.hpp is used to store and calculated robot pose from odometry info.

Class

The **KobukiManager** class contains **kobuki::Kobuki** class, pose data and sigslot callback. **KobukiManager** class is extended from previous **sigslots** examples. Now it has dx, dth and pose data.

```
class KobukiManager {
public:
  KobukiManager() :
    dx(0.0), dth(0.0),
    slot_stream_data(&KobukiManager::processStreamData, *this)
  {
    kobuki::Parameters parameters;
    parameters.sigslots_namespace = "/kobuki";
    parameters.device_port = "/dev/kobuki";
    parameters.enable_acceleration_limiter = false;
    kobuki.init(parameters);
    kobuki.enable();
    slot_stream_data.connect("/kobuki/stream_data");
  }
```

Only **kobuki.enable()** is added from previous sigslot exmaple. It will turn on motor power of kobuki. To moving kobuki, you should call this function before commanding wheel velocities.

```
  ~KobukiManager() {
    kobuki.setBaseControl(0,0); // linear_velocity, angular_velocity in (m/s), (rad/s)
    kobuki.disable();
  }
```

Destructor is added in this example. When quiting the porgam, this code will immediately stop the robot and disable it. It will improve the controllability in emergent condition, and will prevent unwanted behavior.

kobuki.setBaseControl() is a function to send wheel command to robot. It's arguments are linear and angular velocities in (m/s) and (rad/s).

disable() will turned off the motor power.

```
  void processStreamData() {
    ecl::Pose2D<double> pose_update;
    ecl::linear_algebra::Vector3d pose_update_rates;
    kobuki.updateOdometry(pose_update, pose_update_rates);
    pose *= pose_update;
    dx += pose_update.x();
    dth += pose_update.heading();
    processMotion();
  }
```

processStreamData() is slot callback function, when stream data is arrived from robot, this callback will be called.

@ **kobuki::Kobuki::updateOdometry()** method is used to retriving odometry data from last stream data. It returns `updated_pose` and `updated_pose_rate` that calculated from encoder ticks.

pose_update containts calculated linear and angular displacements between the stream. pose_update_rates contains calculated linear and angular velocities as well.

And, by perform below calculation:

```
pose *= pose_update;
```

we can get accumulated pose from odoemtry.

```
  // Generate square motion
  void processMotion() {
    if (dx >= 1.0 && dth >= ecl::pi/2.0) { dx=0.0; dth=0.0; kobuki.setBaseControl(0.0, 0.0);
return; }
    else if (dx >= 1.0) { kobuki.setBaseControl(0.0, 3.3); return; }
    else { kobuki.setBaseControl(0.3, 0.0); return; }
  }
```

processMotion() determines next base command to the robot. These commands will makes robot to follow squre path at the last, by running below commands repeatedly:

- go forward 1.0 meter
- rotate counter-clockwise in 90 degree

```
  ecl::Pose2D<double> getPose() {
    return pose;
  }
```

getPose() is simple getter method for accumulated pose of robot.

```
private:
  double dx, dth;
  ecl::Pose2D<double> pose;
  kobuki::Kobuki kobuki;
  ecl::Slot<> slot_stream_data;
};
```

Signal Handler

```
bool shutdown_req = false;
void signalHandler(int signum) {
  shutdown_req = true;
}
```

This part is signal handler. **signalHandler()** will be called when the SIGINT signal is catched(e.g. user pressed the ctrl+c keys). A bool variable, shutdown_req, setted to false will break the while loop in main function, and terminates program also. If there is no signal handler, program will terminated without calling destructor of **KobukiManager** class. And the robot will run last wheel command continuously until timed out(5 seconds by default).

Main

```
int main(int argc, char** argv)
{
  signal(SIGINT, signalHandler);
```

Call signal() function to install **signalHandler()**.

```
  std::cout << "Demo : Example of simple control loop." << std::endl;
  KobukiManager kobuki_manager;

  ecl::Sleep sleep(1);
  ecl::Pose2D<double> pose;
  try {
    while (!shutdown_req){
      sleep();
      pose = kobuki_manager.getPose();
      std::cout << "current pose: [" << pose.x() << ", " << pose.y() << ", " <<
pose.heading() << "]" << std::endl;
    }
  } catch ( ecl::StandardException &e ) {
    std::cout << e.what();
  }
  return 0;
}
```

Accumulated pose will be printed in every seconds. If you pressed ctrl+c keys, then robot will stop and program will be terminated.