4. Sample Code

Edit on GitHub

4. 1. CPP Protocol 2.0

4. 1. 1. CPP Read Write Protocol 2.0

- Description
  This example writes goal position to the Dynamixel and repeats to read present position until it stops moving. The funtions that are related with the Read and the Write handle the number of items which are near each other in the Dynamixel control table, such as the goal position and the goal velocity.
- Available Dynamixel
  All series using protocol 2.0

4. 1. 1. 1. Sample code

```
/*
 * read_write.cpp
 *
 *  Created on: 2016. 2. 21.
 *      Author: leon
 */

//
// *********     Read and Write Example      *********
//
//
// Available Dynamixel model on this example : All models using Protocol 2.0
// This example is designed for using a Dynamixel PRO 54-200, and an
USB2DYNAMIXEL.
// To use another Dynamixel model, such as X series, see their details in E-
Manual(support.robotis.com) and edit below "#define"d variables yourself.
// Be sure that Dynamixel PRO properties are already set as %% ID : 1 / Baudnum
: 3 (Baudrate : 1000000 [1M])
//

#ifdef __linux__
#include <unistd.h>
#include <fcntl.h>
#include <termios.h>
#elif defined(_WIN32) || defined(_WIN64)
#include <conio.h>
#endif

#include <stdlib.h>
#include <stdio.h>

#include "dynamixel_sdk.h"                                  // Uses Dynamixel
SDK library
```

```
// Control table address
#define ADDR_PRO_TORQUE_ENABLE          562                 // Control table
address is different in Dynamixel model
#define ADDR_PRO_GOAL_POSITION          596
#define ADDR_PRO_PRESENT_POSITION       611

// Protocol version
#define PROTOCOL_VERSION                2.0                 // See which
protocol version is used in the Dynamixel

// Default setting
#define DXL_ID                          1                   // Dynamixel ID: 1
#define BAUDRATE                        1000000
#define DEVICENAME                      "/dev/ttyUSB0"      // Check which port
is being used on your controller
                                                            // ex) Windows:
"COM1"   Linux: "/dev/ttyUSB0"

#define TORQUE_ENABLE                   1                   // Value for
enabling the torque
#define TORQUE_DISABLE                  0                   // Value for
disabling the torque
#define DXL_MINIMUM_POSITION_VALUE      -150000             // Dynamixel will
rotate between this value
#define DXL_MAXIMUM_POSITION_VALUE      150000              // and this value
(note that the Dynamixel would not move when the position value is out of
movable range. Check e-manual about the range of the Dynamixel you use.)
#define DXL_MOVING_STATUS_THRESHOLD     20                  // Dynamixel moving
status threshold

#define ESC_ASCII_VALUE                 0x1b

int getch()
{
#ifdef __linux__
  struct termios oldt, newt;
  int ch;
  tcgetattr(STDIN_FILENO, &oldt);
  newt = oldt;
  newt.c_lflag &= ~(ICANON | ECHO);
  tcsetattr(STDIN_FILENO, TCSANOW, &newt);
  ch = getchar();
  tcsetattr(STDIN_FILENO, TCSANOW, &oldt);
  return ch;
#elif defined(_WIN32) || defined(_WIN64)
  return _getch();
#endif
}

int kbhit(void)
```

```
{
#ifdef __linux__
  struct termios oldt, newt;
  int ch;
  int oldf;

  tcgetattr(STDIN_FILENO, &oldt);
  newt = oldt;
  newt.c_lflag &= ~(ICANON | ECHO);
  tcsetattr(STDIN_FILENO, TCSANOW, &newt);
  oldf = fcntl(STDIN_FILENO, F_GETFL, 0);
  fcntl(STDIN_FILENO, F_SETFL, oldf | O_NONBLOCK);

  ch = getchar();

  tcsetattr(STDIN_FILENO, TCSANOW, &oldt);
  fcntl(STDIN_FILENO, F_SETFL, oldf);

  if (ch != EOF)
  {
    ungetc(ch, stdin);
    return 1;
  }

  return 0;
#elif defined(_WIN32) || defined(_WIN64)
  return _kbhit();
#endif
}

int main()
{
  // Initialize PortHandler instance
  // Set the port path
  // Get methods and members of PortHandlerLinux or PortHandlerWindows
  dynamixel::PortHandler *portHandler =
dynamixel::PortHandler::getPortHandler(DEVICENAME);

  // Initialize PacketHandler instance
  // Set the protocol version
  // Get methods and members of Protocol1PacketHandler or Protocol2PacketHandler
  dynamixel::PacketHandler *packetHandler =
dynamixel::PacketHandler::getPacketHandler(PROTOCOL_VERSION);

  int index = 0;
  int dxl_comm_result = COMM_TX_FAIL;             // Communication result
  int dxl_goal_position[2] = {DXL_MINIMUM_POSITION_VALUE,
DXL_MAXIMUM_POSITION_VALUE};         // Goal position

  uint8_t dxl_error = 0;                          // Dynamixel error
  int32_t dxl_present_position = 0;               // Present position
```

```
  // Open port
  if (portHandler->openPort())
  {
    printf("Succeeded to open the port!\n");
  }
  else
  {
    printf("Failed to open the port!\n");
    printf("Press any key to terminate...\n");
    getch();
    return 0;
  }

  // Set port baudrate
  if (portHandler->setBaudRate(BAUDRATE))
  {
    printf("Succeeded to change the baudrate!\n");
  }
  else
  {
    printf("Failed to change the baudrate!\n");
    printf("Press any key to terminate...\n");
    getch();
    return 0;
  }

  // Enable Dynamixel Torque
  dxl_comm_result = packetHandler->write1ByteTxRx(portHandler, DXL_ID,
ADDR_PRO_TORQUE_ENABLE, TORQUE_ENABLE, &dxl_error);
  if (dxl_comm_result != COMM_SUCCESS)
  {
    packetHandler->printTxRxResult(dxl_comm_result);
  }
  else if (dxl_error != 0)
  {
    packetHandler->printRxPacketError(dxl_error);
  }
  else
  {
    printf("Dynamixel has been successfully connected \n");
  }

  while(1)
  {
    printf("Press any key to continue! (or press ESC to quit!)\n");
    if (getch() == ESC_ASCII_VALUE)
      break;

    // Write goal position
    dxl_comm_result = packetHandler->write4ByteTxRx(portHandler, DXL_ID,
```

```
ADDR_PRO_GOAL_POSITION, dxl_goal_position[index], &dxl_error);
    if (dxl_comm_result != COMM_SUCCESS)
    {
      packetHandler->printTxRxResult(dxl_comm_result);
    }
    else if (dxl_error != 0)
    {
      packetHandler->printRxPacketError(dxl_error);
    }

    do
    {
      // Read present position
      dxl_comm_result = packetHandler->read4ByteTxRx(portHandler, DXL_ID,
ADDR_PRO_PRESENT_POSITION, (uint32_t*)&dxl_present_position, &dxl_error);
      if (dxl_comm_result != COMM_SUCCESS)
      {
        packetHandler->printTxRxResult(dxl_comm_result);
      }
      else if (dxl_error != 0)
      {
        packetHandler->printRxPacketError(dxl_error);
      }

      printf("[ID:%03d] GoalPos:%03d  PresPos:%03d\n", DXL_ID,
dxl_goal_position[index], dxl_present_position);

    }while((abs(dxl_goal_position[index] - dxl_present_position) >
DXL_MOVING_STATUS_THRESHOLD));

    // Change goal position
    if (index == 0)
    {
      index = 1;
    }
    else
    {
      index = 0;
    }
  }

  // Disable Dynamixel Torque
  dxl_comm_result = packetHandler->write1ByteTxRx(portHandler, DXL_ID,
ADDR_PRO_TORQUE_ENABLE, TORQUE_DISABLE, &dxl_error);
  if (dxl_comm_result != COMM_SUCCESS)
  {
    packetHandler->printTxRxResult(dxl_comm_result);
  }
  else if (dxl_error != 0)
  {
    packetHandler->printRxPacketError(dxl_error);
```

```
  }

  // Close port
  portHandler->closePort();

  return 0;
}
```

4. 1. 1. 2. Details

```
#ifdef __linux__
#include <unistd.h>
#include <fcntl.h>
#include <termios.h>
#elif defined(_WIN32) || defined(_WIN64)
#include <conio.h>
#endif
```

This source includes above to get key input interruption while the example is running. Actual functions for getting the input is described in a little below.

```
#include <stdlib.h>
```

The function `abs()` is in the example code, and it needs `stdlib.h` to be included.

```
#include <stdio.h>
```

The example shows Dynamixel status in sequence by the function `printf()`. So here `stdio.h` is needed.

```
#include "dynamixel_sdk.h"                                  // Uses Dynamixel
SDK library
```

All libraries of Dynamixel SDK are linked with the header file `dynamixel_sdk.h`.

```
// Control table address
#define ADDR_PRO_TORQUE_ENABLE          562                 // Control table
address is different in Dynamixel model
#define ADDR_PRO_GOAL_POSITION          596
#define ADDR_PRO_PRESENT_POSITION       611
```

Dynamixel series have their own control tables: Addresses and Byte Length in each items. To control one of the items, its address (and length if necessary) is required. Find your requirements in http://support.robotis.com/.

```
// Protocol version
#define PROTOCOL_VERSION                2.0                 // See which
protocol version is used in the Dynamixel
```

Dynamixel uses either or both protocols: Protocol 1.0 and Protocol 2.0. Choose one of the Protocol which is appropriate in the Dynamixel.

```
// Default setting
#define DXL_ID                          1                   // Dynamixel ID: 1
#define BAUDRATE                        1000000
#define DEVICENAME                      "/dev/ttyUSB0"      // Check which port
is being used on your controller
                                                            // ex) Windows:
"COM1"   Linux: "/dev/ttyUSB0"

#define TORQUE_ENABLE                   1                   // Value for
enabling the torque
#define TORQUE_DISABLE                  0                   // Value for
disabling the torque
#define DXL_MINIMUM_POSITION_VALUE      -150000             // Dynamixel will
rotate between this value
#define DXL_MAXIMUM_POSITION_VALUE      150000              // and this value
(note that the Dynamixel would not move when the position value is out of
movable range. Check e-manual about the range of the Dynamixel you use.)
#define DXL_MOVING_STATUS_THRESHOLD     20                  // Dynamixel moving
status threshold


#define ESC_ASCII_VALUE                 0x1b
```

Here we set some variables to let you freely change them and use them to run the example code.

As the document already said in previous chapter, customize Dynamixel control table items, such as `DXL_ID` number, communication `BAUDRATE`, and the `DEVICENAME`, on your own terms of needs. In particular, `BAUDRATE` and `DEVICENAME` have systematical dependencies on your controller, so make clear what kind of communication method you will use.

Dynamixel basically needs the `TORQUE_ENABLE` to be rotating or give you its internal information. On the other hand, it doesn't need torque enabled if you get your goal, so finally do `TORQUE_DISABLE` to prepare to the next sequence.

Since the Dynamixel has its own rotation range, it may shows malfunction if your request on your dynamixel is out of range. For example, Dynamixel MX-28 and Dynamixel PRO 54-200 has its rotatable range as 0 ~ 4028 and -250950 ~ 250950, each.

`DXL_MOVING_STATUS_THRESHOLD` acts as a criteria for verifying its rotation stopped.

```
int getch()
{
#ifdef __linux__
  struct termios oldt, newt;
  int ch;
  tcgetattr(STDIN_FILENO, &oldt);
  newt = oldt;
  newt.c_lflag &= ~(ICANON | ECHO);
  tcsetattr(STDIN_FILENO, TCSANOW, &newt);
  ch = getchar();
  tcsetattr(STDIN_FILENO, TCSANOW, &oldt);
  return ch;
#elif defined(_WIN32) || defined(_WIN64)
  return _getch();
```

```
#endif
}

int kbhit(void)
{
#ifdef __linux__
  struct termios oldt, newt;
  int ch;
  int oldf;

  tcgetattr(STDIN_FILENO, &oldt);
  newt = oldt;
  newt.c_lflag &= ~(ICANON | ECHO);
  tcsetattr(STDIN_FILENO, TCSANOW, &newt);
  oldf = fcntl(STDIN_FILENO, F_GETFL, 0);
  fcntl(STDIN_FILENO, F_SETFL, oldf | O_NONBLOCK);

  ch = getchar();

  tcsetattr(STDIN_FILENO, TCSANOW, &oldt);
  fcntl(STDIN_FILENO, F_SETFL, oldf);

  if (ch != EOF)
  {
    ungetc(ch, stdin);
    return 1;
  }

  return 0;
#elif defined(_WIN32) || defined(_WIN64)
  return _kbhit();
#endif
}
```

These functions accept the key inputs in terms of example action. The example codes mainly apply the function `getch()` rather than the function `kbhit()` to get information which key has been pressed.

```
int main()
{
  // Initialize PortHandler instance
  // Set the port path
  // Get methods and members of PortHandlerLinux or PortHandlerWindows
  dynamixel::PortHandler *portHandler =
dynamixel::PortHandler::getPortHandler(DEVICENAME);

  // Initialize PacketHandler instance
  // Set the protocol version
  // Get methods and members of Protocol1PacketHandler or Protocol2PacketHandler
  dynamixel::PacketHandler *packetHandler =
dynamixel::PacketHandler::getPacketHandler(PROTOCOL_VERSION);
```

```
  int index = 0;
  int dxl_comm_result = COMM_TX_FAIL;             // Communication result
  int dxl_goal_position[2] = {DXL_MINIMUM_POSITION_VALUE,
DXL_MAXIMUM_POSITION_VALUE};         // Goal position

  uint8_t dxl_error = 0;                          // Dynamixel error
  int32_t dxl_present_position = 0;               // Present position

  // Open port
  if (portHandler->openPort())
  {
    printf("Succeeded to open the port!\n");
  }
  else
  {
    printf("Failed to open the port!\n");
    printf("Press any key to terminate...\n");
    getch();
    return 0;
  }

  // Set port baudrate
  if (portHandler->setBaudRate(BAUDRATE))
  {
    printf("Succeeded to change the baudrate!\n");
  }
  else
  {
    printf("Failed to change the baudrate!\n");
    printf("Press any key to terminate...\n");
    getch();
    return 0;
  }

  // Enable Dynamixel Torque
  dxl_comm_result = packetHandler->write1ByteTxRx(portHandler, DXL_ID,
ADDR_PRO_TORQUE_ENABLE, TORQUE_ENABLE, &dxl_error);
  if (dxl_comm_result != COMM_SUCCESS)
  {
    packetHandler->printTxRxResult(dxl_comm_result);
  }
  else if (dxl_error != 0)
  {
    packetHandler->printRxPacketError(dxl_error);
  }
  else
  {
    printf("Dynamixel has been successfully connected \n");
  }

  while(1)
```

```
  {
    printf("Press any key to continue! (or press ESC to quit!)\n");
    if (getch() == ESC_ASCII_VALUE)
      break;

    // Write goal position
    dxl_comm_result = packetHandler->write4ByteTxRx(portHandler, DXL_ID,
ADDR_PRO_GOAL_POSITION, dxl_goal_position[index], &dxl_error);
    if (dxl_comm_result != COMM_SUCCESS)
    {
      packetHandler->printTxRxResult(dxl_comm_result);
    }
    else if (dxl_error != 0)
    {
      packetHandler->printRxPacketError(dxl_error);
    }

    do
    {
      // Read present position
      dxl_comm_result = packetHandler->read4ByteTxRx(portHandler, DXL_ID,
ADDR_PRO_PRESENT_POSITION, (uint32_t*)&dxl_present_position, &dxl_error);
      if (dxl_comm_result != COMM_SUCCESS)
      {
        packetHandler->printTxRxResult(dxl_comm_result);
      }
      else if (dxl_error != 0)
      {
        packetHandler->printRxPacketError(dxl_error);
      }

      printf("[ID:%03d] GoalPos:%03d  PresPos:%03d\n", DXL_ID,
dxl_goal_position[index], dxl_present_position);

    }while((abs(dxl_goal_position[index] - dxl_present_position) >
DXL_MOVING_STATUS_THRESHOLD));

    // Change goal position
    if (index == 0)
    {
      index = 1;
    }
    else
    {
      index = 0;
    }
  }

  // Disable Dynamixel Torque
  dxl_comm_result = packetHandler->write1ByteTxRx(portHandler, DXL_ID,
ADDR_PRO_TORQUE_ENABLE, TORQUE_DISABLE, &dxl_error);
```

```
  if (dxl_comm_result != COMM_SUCCESS)
  {
    packetHandler->printTxRxResult(dxl_comm_result);
  }
  else if (dxl_error != 0)
  {
    packetHandler->printRxPacketError(dxl_error);
  }

  // Close port
  portHandler->closePort();

  return 0;
}
```

In `main()` function, the codes call actual functions for Dynamixel control.

```
  // Initialize PortHandler instance
  // Set the port path
  // Get methods and members of PortHandlerLinux or PortHandlerWindows
  dynamixel::PortHandler *portHandler =
dynamixel::PortHandler::getPortHandler(DEVICENAME);
```

`getPortHandler()` function sets port path as `DEVICENAME`, and prepare an appropriate `dynamixel::PortHandler` in controller OS automatically.

```
  // Initialize PacketHandler instance
  // Set the protocol version
  // Get methods and members of Protocol1PacketHandler or Protocol2PacketHandler
  dynamixel::PacketHandler *packetHandler =
dynamixel::PacketHandler::getPacketHandler(PROTOCOL_VERSION);
```

`getPacketHandler()` function sets the methods for packet construction by choosing the `PROTOCOL_VERSION`.

```
  int index = 0;
  int dxl_comm_result = COMM_TX_FAIL;             // Communication result
  int dxl_goal_position[2] = {DXL_MINIMUM_POSITION_VALUE,
DXL_MAXIMUM_POSITION_VALUE};         // Goal position

  uint8_t dxl_error = 0;                          // Dynamixel error
  int32_t dxl_present_position = 0;               // Present position
```

`index` variable points the direction to where the Dynamixel should be rotated.

`dxl_comm_result` indicates which error has been occurred during packet communication.

`dxl_goal_position` stores goal points of Dynamixel rotation.

`dxl_error` shows the internal error in Dynamixel.

`dxl_present_position` views where now it points out.

```
  // Open port
```

```
  if (portHandler->openPort())
  {
    printf("Succeeded to open the port!\n");
  }
  else
  {
    printf("Failed to open the port!\n");
    printf("Press any key to terminate...\n");
    getch();
    return 0;
  }
```

First, controller opens the port to do serial communication with the Dynamixel. If it fails to open the port, the example will be terminated.

```
  // Set port baudrate
  if (portHandler->setBaudRate(BAUDRATE))
  {
    printf("Succeeded to change the baudrate!\n");
  }
  else
  {
    printf("Failed to change the baudrate!\n");
    printf("Press any key to terminate...\n");
    getch();
    return 0;
  }
```

Secondly, the controller sets the communication `BAUDRATE` at the port opened previously.

```
  // Enable Dynamixel Torque
  dxl_comm_result = packetHandler->write1ByteTxRx(portHandler, DXL_ID,
ADDR_PRO_TORQUE_ENABLE, TORQUE_ENABLE, &dxl_error);
  if (dxl_comm_result != COMM_SUCCESS)
  {
    packetHandler->printTxRxResult(dxl_comm_result);
  }
  else if (dxl_error != 0)
  {
    packetHandler->printRxPacketError(dxl_error);
  }
  else
  {
    printf("Dynamixel has been successfully connected \n");
  }
```

As mentioned in the document, above code enables Dynamixel torque to set its status as being ready to move.

`dynamixel::PacketHandler::write1ByteTxRx()` function orders to the #`DXL_ID` Dynamixel through the port which the `portHandler` handles, writing 1 byte of `TORQUE_ENABLE` value to `ADDR_PRO_TORQUE_ENABLE` address. Then, it receives the `dxl_error`. The function returns 0 if no communication error has been occurred.

```
  while(1)
  {
    printf("Press any key to continue! (or press ESC to quit!)\n");
    if (getch() == ESC_ASCII_VALUE)
      break;

    // Write goal position
    dxl_comm_result = packetHandler->write4ByteTxRx(portHandler, DXL_ID,
ADDR_PRO_GOAL_POSITION, dxl_goal_position[index], &dxl_error);
    if (dxl_comm_result != COMM_SUCCESS)
    {
      packetHandler->printTxRxResult(dxl_comm_result);
    }
    else if (dxl_error != 0)
    {
      packetHandler->printRxPacketError(dxl_error);
    }

    do
    {
      // Read present position
      dxl_comm_result = packetHandler->read4ByteTxRx(portHandler, DXL_ID,
ADDR_PRO_PRESENT_POSITION, (uint32_t*)&dxl_present_position, &dxl_error);
      if (dxl_comm_result != COMM_SUCCESS)
      {
        packetHandler->printTxRxResult(dxl_comm_result);
      }
      else if (dxl_error != 0)
      {
        packetHandler->printRxPacketError(dxl_error);
      }

      printf("[ID:%03d] GoalPos:%03d  PresPos:%03d\n", DXL_ID,
dxl_goal_position[index], dxl_present_position);

    }while((abs(dxl_goal_position[index] - dxl_present_position) >
DXL_MOVING_STATUS_THRESHOLD));

    // Change goal position
    if (index == 0)
    {
      index = 1;
    }
    else
    {
      index = 0;
    }
  }

  // Disable Dynamixel Torque
```

```
  dxl_comm_result = packetHandler->write1ByteTxRx(portHandler, DXL_ID,
ADDR_PRO_TORQUE_ENABLE, TORQUE_DISABLE, &dxl_error);
  if (dxl_comm_result != COMM_SUCCESS)
  {
    packetHandler->printTxRxResult(dxl_comm_result);
  }
  else if (dxl_error != 0)
  {
    packetHandler->printRxPacketError(dxl_error);
  }

  // Close port
  portHandler->closePort();

  return 0;
}
```

During `while()` loop, the controller writes and reads the Dynamixel position through packet transmission/reception(Tx/Rx).

To continue its rotation, press any key except ESC.

`dynamixel::PacketHandler::write4ByteTxRx()` function orders to the #`DXL_ID` Dynamixel through the port which the `portHandler` handles, writing 4 bytes of `dxl_goal_position[index]` value to `ADDR_PRO_GOAL_POSITION` address. Then, it receives the `dxl_error`. The function returns 0 if no communication error has been occurred.

`dynamixel::PacketHandler::read4ByteTxRx()` functions orders to the #`DXL_ID` Dynamixel through the port which the `portHandler` handles, requesting 4 bytes of value of `ADDR_PRO_PRESENT_POSITION` address. Then, it receives `dxl_present_position` and `dxl_error`. The function returns 0 if no communication error has been occurred.

Reading its present position will be ended when absolute value of `(dxl_goal_position[index] - dxl_present_position)` becomes smaller then `DXL_MOVING_STATUS_THRESHOLD`.

At last, it changes its direction to the counter-wise and waits for extra key input.

```
  // Disable Dynamixel Torque
  dxl_comm_result = packetHandler->write1ByteTxRx(portHandler, DXL_ID,
ADDR_PRO_TORQUE_ENABLE, TORQUE_DISABLE, &dxl_error);
  if (dxl_comm_result != COMM_SUCCESS)
  {
    packetHandler->printTxRxResult(dxl_comm_result);
  }
  else if (dxl_error != 0)
  {
    packetHandler->printRxPacketError(dxl_error);
  }
```

The controller frees the Dynamixel to be idle.

`dynamixel::PacketHandler::write1ByteTxRx()` function orders to the #`DXL_ID` Dynamixel through the port which the `portHandler` handles, writing 1 byte of `TORQUE_DISABLE` value to `ADDR_PRO_TORQUE_ENABLE` address. Then, it receives the `dxl_error`. The function returns 0 if no communication error has been occurred.

```
// Close port
portHandler->closePort();

return 0;
```

Finally, port becomes disposed.