Rollo - Localization of a humanoid robot

1.0

Generated by Doxygen 1.8.10

Fri Mar 4 2016 10:41:18

Contents

Chapter 1

Main Page

University of Genova

86805 - Software Architectures for Robotics

Assignment 03 - Localization system for a wheeled humanoid robot

Author

Rabbia Asghar
Ernest Skrzypczyk

Project: Rollo - Localization of a humanoid robot

Summary: Project deals with the aspect of localization of a humanoid robot using the extended Kalman filter and ROS as working environment with help of motion capture.

Description: Given was the humanoid robot Rollo with a mechanical and electrical construction that enabled reesembling human walking motion.

Goal: Implementation of localization of Rollo using ROS environment and extended Kalman filter with motion capture data as one of the information sources.

Implemented work

Tasks performed for the localization of Rollo using a ROS environment with the help of a motion capture system included:

- Modelling of the physical system of the robot
- Taking into consideration odometry errors: systematic and non-systematic
- Performing the square test based on UMBMark
- Designing the system and measurement models
- Implementing the extended Kalman filter into ROS
- Developing all necessary nodes to control and communicate with Rollo using ROS
- Developing nodes for visualization and preprocessing of motion capture data

Possible improvements and future work

The extended Kalman filter for localization has been successfully implemented. There is still room for improvement, especially with the odometry model, but that requires more intesive testing on the robot. Furthermore the ROS nodes could be improved:

- Add process and measurement covariance matrices into the header file rollo.hpp
- Visualization node could also provide analysis of the EKF performance
- Visualization node could be greatly expanded for media purposes by fully utilizing already provided code like saving animations and screenshots
- EKF node could reinitialize the odometry model at a given refresh period, should the model be incomplete
- Implementation of dynamic reconfiguration server into the ROS nodes, would greatly improve work with prototypes and changing environments

Further changes on the Rollo should include:

- Use of better connectors in the power circuits is necessary, even though wire connections have been greatly improved already
- Provide at least a basic shortage protection circuit: This can be done using diodes, like 1N4007, on the input and output of the voltage regulators; One in series to rectify and one in parallel to protect in case of opposite polarities
- Exchange and calibration of wheels is necessary
- A different material for the wheel surface might be an asset

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

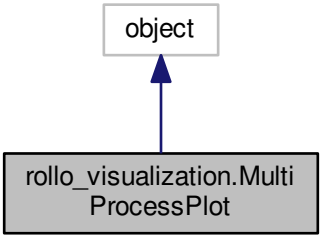
Here is a list of all documented files with brief descriptions:

Chapter 5

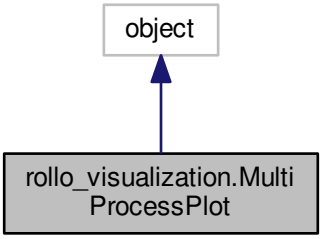
Class Documentation

5.1 rollo_visualization.MultiProcessPlot Class Reference

Inheritance diagram for rollo_visualization.MultiProcessPlot:



Collaboration diagram for rollo_visualization.MultiProcessPlot:



Public Member Functions

- def __init__ (self)

Initilization.

- def plot

  Plot function.

Public Attributes

- plotter

  Called process for plotting.

- plotprocess

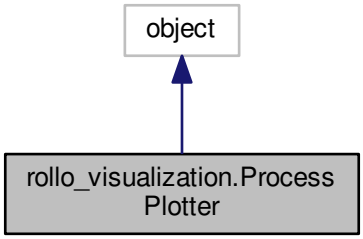
  Process holder.

The documentation for this class was generated from the following file:

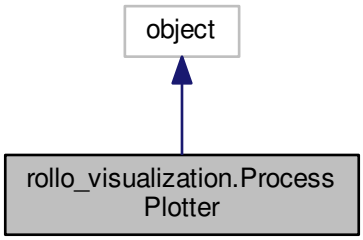
- rollo_visualization.py

5.2 rollo_visualization.ProcessPlotter Class Reference

Inheritance diagram for rollo_visualization.ProcessPlotter:



Collaboration diagram for rollo_visualization.ProcessPlotter:



Public Member Functions

- def **__init__** (self)
- def **terminate** (self)
- def **poll_draw** (self)
- def **__call__** (self, pipe)

Public Attributes

- x1

  X data from motion capture.
- y1

  Y data from motion capture.
- x2

  X data from odometry model.
- y2

  Y data from odometry model.
- x3

  X data from EKF estimates.
- y3

  Y data from EKF estimates.
- pipe

  Data transmission pipe between processes.
- **ax**
- **gid**

The documentation for this class was generated from the following file:

- rollo_visualization.py

5.3 udp_client_server::udp_client Class Reference

Public Member Functions

- udp_client (const std::string &addr, int port)

  Initialize a UDP client object.
- ∼udp_client ()

  Clean up the UDP client object.
- int get_socket () const

  Retrieve a copy of the socket identifier.
- int get_port () const

  Retrieve the port used by this UDP client.
- std::string get_addr () const

  Retrieve a copy of the address.
- int send (const char ∗msg, size_t size)

  Send a message through this UDP client.

5.3.1 Constructor & Destructor Documentation

5.3.1.1 udp_client_server::udp_client::udp_client (const std::string & *addr,* int *port*)

Initialize a UDP client object.

This function initializes the UDP client object using the address and the port as specified.

The port is expected to be a host side port number (i.e. 59200).

The `addr` parameter is a textual address. It may be an IPv4 or IPv6 address and it can represent a host name or an address defined with just numbers. If the address cannot be resolved then an error occurs and constructor throws.

Note

The socket is open in this process. If you fork() or exec() then the socket will be closed by the operating system.

Warning

We only make use of the first address found by getaddrinfo(). All the other addresses are ignored.

Exceptions

udp_client_server_↩ runtime_error	The server could not be initialized properly. Either the address cannot be resolved, the port is incompatible or not available, or the socket could not be created.

Parameters

`in`	*addr*	The address to convert to a numeric IP.
`in`	*port*	The port number.

5.3.1.2 udp_client_server::udp_client::∼udp_client ( )

Clean up the UDP client object.

This function frees the address information structure and close the socket before returning.

5.3.2 Member Function Documentation

5.3.2.1 std::string udp_client_server::udp_client::get_addr ( ) const

Retrieve a copy of the address.

This function returns a copy of the address as it was specified in the constructor. This does not return a canonalized version of the address.

The address cannot be modified. If you need to send data on a different address, create a new UDP client.

Returns

A string with a copy of the constructor input address.

5.3.2.2 int udp_client_server::udp_client::get_port ( ) const

Retrieve the port used by this UDP client.

This function returns the port used by this UDP client. The port is defined as an integer, host side.

Returns

The port as expected in a host integer.

5.3.2.3 int udp_client_server::udp_client::get_socket ( ) const

Retrieve a copy of the socket identifier.

This function return the socket identifier as returned by the socket() function. This can be used to change some flags.

Returns

The socket used by this UDP client.

5.3.2.4 int udp_client_server::udp_client::send (const char * *msg*, size_t *size*)

Send a message through this UDP client.

This function sends `msg` through the UDP client socket. The function cannot be used to change the destination as it was defined when creating the udp_client object.

The size must be small enough for the message to fit. In most cases we use these in Snap! to send very small signals (i.e. 4 bytes commands.) Any data we would want to share remains in the Cassandra database so that way we can avoid losing it because of a UDP message.

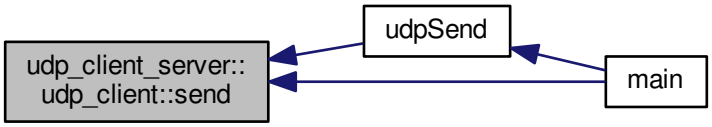
Parameters

`in`	*msg*	The message to send.
`in`	*size*	The number of bytes representing this message.

Returns

-1 if an error occurs, otherwise the number of bytes sent. errno is set accordingly on error.

Here is the caller graph for this function:

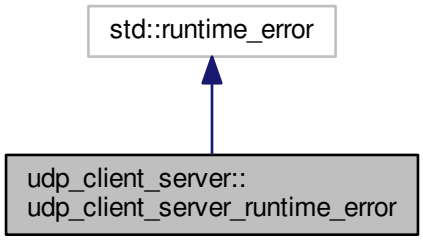


The documentation for this class was generated from the following files:

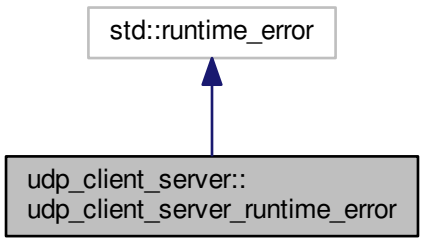
- udp.h
- udp.cpp

5.4 udp_client_server::udp_client_server_runtime_error Class Reference

Inheritance diagram for udp_client_server::udp_client_server_runtime_error:



Collaboration diagram for udp_client_server::udp_client_server_runtime_error:



Public Member Functions

- **udp_client_server_runtime_error** (const char ∗w)

The documentation for this class was generated from the following file:

- udp.h

5.5 udp_client_server::udp_server Class Reference

Public Member Functions

- udp_server (const std::string &addr, int port)

  Initialize a UDP server object.
- ∼udp_server ()

  Clean up the UDP server.

- int get_socket () const

  The socket used by this UDP server.
- int get_port () const

  The port used by this UDP server.
- std::string get_addr () const

  Return the address of this UDP server.
- int recv (char *msg, size_t max_size)

  Wait on a message.
- int timed_recv (char *msg, size_t max_size, int max_wait_ms)

  Wait for data to come in.

5.5.1 Constructor & Destructor Documentation

5.5.1.1 udp_client_server::udp_server::udp_server (const std::string & *addr*, int *port*)

Initialize a UDP server object.

This function initializes a UDP server object making it ready to receive messages.

The server address and port are specified in the constructor so if you need to receive messages from several different addresses and/or port, you'll have to create a server for each.

The address is a string and it can represent an IPv4 or IPv6 address.

Note that this function calls connect() to connect the socket to the specified address. To accept data on different UDP addresses and ports, multiple UDP servers must be created.

Note

The socket is open in this process. If you fork() or exec() then the socket will be closed by the operating system.

Warning

We only make use of the first address found by getaddrinfo(). All the other addresses are ignored.

Exceptions

udp_client_server_runtime_error	The udp_client_server_runtime_error exception is raised when the address and port combinaison cannot be resolved or if the socket cannot be opened.

Parameters

in	*addr*	The address we receive on.
in	*port*	The port we receive from.

5.5.1.2 udp_client_server::udp_server::~udp_server ( )

Clean up the UDP server.

This function frees the address info structures and close the socket.

5.5.2 Member Function Documentation

5.5.2.1 std::string udp_client_server::udp_server::get_addr ( ) const

Return the address of this UDP server.

This function returns a verbatim copy of the address as passed to the constructor of the UDP server (i.e. it does not return the canonalized version of the address.)

Returns

The address as passed to the constructor.

5.5.2.2 int udp_client_server::udp_server::get_port ( ) const

The port used by this UDP server.

This function returns the port attached to the UDP server. It is a copy of the port specified in the constructor.

Returns

The port of the UDP server.

5.5.2.3 int udp_client_server::udp_server::get_socket ( ) const

The socket used by this UDP server.

This function returns the socket identifier. It can be useful if you are doing a select() on many sockets.

Returns

The socket of this UDP server.

5.5.2.4 int udp_client_server::udp_server::recv (char * *msg*, size_t *max_size*)

Wait on a message.

This function waits until a message is received on this UDP server. There are no means to return from this function except by receiving a message. Remember that UDP does not have a connect state so whether another process quits does not change the status of this UDP server and thus it continues to wait forever.

Note that you may change the type of socket by making it non-blocking (use the get_socket() to retrieve the socket identifier) in which case this function will not block if no message is available. Instead it returns immediately.

Parameters

in	*msg*	The buffer where the message is saved.
in	*max_size*	The maximum size the message (i.e. size of the `msg` buffer.)

Returns

The number of bytes read or -1 if an error occurs.

5.5.2.5 int udp_client_server::udp_server::timed_recv (char * *msg*, size_t *max_size*, int *max_wait_ms*)

Wait for data to come in.

This function waits for a given amount of time for data to come in. If no data comes in after max_wait_ms, the function returns with -1 and errno set to EAGAIN.

The socket is expected to be a blocking socket (the default,) although it is possible to setup the socket as non-blocking if necessary for some other reason.

This function blocks for a maximum amount of time as defined by max_wait_ms. It may return sooner with an error or a message.

Parameters

in	*msg*	The buffer where the message will be saved.
in	*max_size*	The size of the `msg` buffer in bytes.
in	*max_wait_ms*	The maximum number of milliseconds to wait for a message.

Returns

-1 if an error occurs or the function timed out, the number of bytes received otherwise.

The documentation for this class was generated from the following files:

- udp.h
- udp.cpp

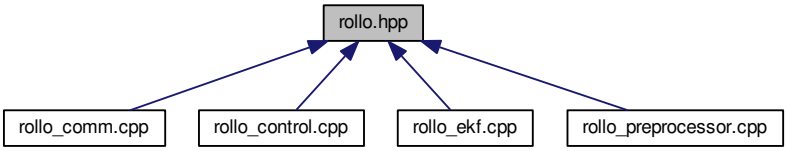
Chapter 6

File Documentation

6.1 rollo.hpp File Reference

Header file holding Rollo specific parameters and global references for the ROS nodes.

This graph shows which files directly or indirectly include this file:



Macros

- #define ROLLO_AXLE_L 0.0205

  Axle length.
- #define ROLLO_WHEEL_RADIUS_L 0.0076

  Wheel left radius.
- #define ROLLO_WHEEL_RADIUS_R 0.0076

  Wheel right radius.
- #define ROLLO_WHEEL_N 4

  Number of wheels.
- #define ROLLO_SPEED_MAX 56

  Maximum speed [%].
- #define ROLLO_SPEED_MIN 6

  Minimum speed [%].
- #define PI 3.1415926535

  Pi.
- #define CM "COMM"

  Communication.
- #define CT "CTRL"

  Control.
- #define LC "LOC "

Localization.

- #define OD "ODOM"

  Odometry.

- #define PP "PREP"

  Preprocessor.

- #define KF "EKF "

  Extended Kalman filter.

- #define VS "VIS "

  Visualization.

- #define PACKAGE "Rollo"

  ROS package name.

- #define TOPIC_COMM_WS "/Rollo/wheelspeed"

  Topic for wheel speed containing the actual speed of wheel, preferably extracted from encoders or if not available by using a lookup table.

- #define TOPIC_CTRL_CMD_VEL "/Rollo/cmd_vel"

  Topic for commands generated by control node expressed in linear and angular velocity.

- #define TOPIC_EKF "/Rollo/ekf"

  Topic for extended Kalman filter results with all three estimated states and covariance matrix, stamped.

- #define TOPIC_PREP_MC "/Optitrack_Rollo/ground_pose"

  Topic for motion capture data.

- #define TOPIC_PREP_P2DT "/Rollo/pose2dstamped"

  Topic for position and orientation, stamped.

- #define CR "\033[0m"

  Reset.

- #define C1 "\033[38;5;63m"

  Color 1 - #5F5FFE.

- #define C2 "\033[38;5;220m"

  Color 2 - #FFD700.

- #define C3 "\033[38;5;87m"

  Color 3 - #5FFFFF.

- #define C4 "\033[38;5;84m"

  Color 4 - #5FFF86.

- #define C5 "\033[38;5;160m"

  Color 5 - #790303.

- #define C6 "\033[38;5;161m"

  Color 6 - #D7005F.

- #define C7 "\033[38;5;162m"

  Color 7 - #D70080.

- #define C8 "\033[38;5;22m"

  Color 8 - #005F00.

- #define CEE "\033[38;5;124m" /∗ Error ∗/

  Error.

- #define CSS "\033[38;5;154m" /∗ Success ∗/

  Success.

- #define CWW "\033[38;5;202m" /∗ Warning ∗/

  Warning.

6.1.1 Detailed Description

Header file holding Rollo specific parameters and global references for the ROS nodes.

Author

Rabbia Asghar
Ernest Skrzypczyk

Date

20/2/16

Project github repository

See also

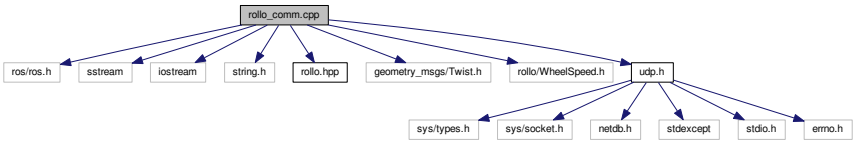
https://github.com/em-er-es/rollo/

6.2 rollo_comm.cpp File Reference

Communication between ROS and Rollo.

```
#include "ros/ros.h"
#include <sstream>
#include <iostream>
#include "string.h"
#include "rollo.hpp"
#include "geometry_msgs/Twist.h"
#include "rollo/WheelSpeed.h"
#include "udp.h"
```

Include dependency graph for rollo_comm.cpp:



Functions

- int decodeVelocities (double x, double z, char ∗Message, int &VelocityL, int &VelocityR)

  Decode linear and angular velocities.
- void EstimateFeedbackVelocities (int VelocityL, int VelocityR, double &LeftVelocityEstimate, double &Right↩ VelocityEstimate)

  Assign left and right wheel velocity estimates for a given velocity command.
- void subscriberCallback (const geometry_msgs::Twist::ConstPtr &msg)

  Subscriber callback from control node.
- int udpSend (char ip[16], int port, char ∗Message)

  Send UDP packets.
- int main (int argc, char ∗∗argv)

  Node main.

Variables

- char NodeName [20] = C3 CM CR

  Node name using console codes.
- char TopicWheelSpeed [64] = TOPIC_COMM_WS

  Topic for wheel speed containing the actual speed of wheel, preferably extracted from encoders or if not available by using a lookup table.
- char TopicCmdVel [64] = TOPIC_CTRL_CMD_VEL

  Topic for commands generated by control node expressed in linear and angular velocity.
- char ip [16] = "192.168.0.120"

  Rollo ip address hardcoded.
- int port = 900

  UDP port.
- double tol = 0.01

  Tolerance for determining linear and angular velocities from the control node.
- int v_l

  Velocity for left wheel.
- int v_r

  Velocity for right wheel.
- unsigned const int nb = 3

  Number of bytes in the message.
- char Message [nb] = {0x7b, 0x50, 0x10}

  Message combined, complete stop default.
- char MessageEmergencyStop [nb] = {0x7b, 0x50, 0x10}

  Emergency message - complete stop.
- double lastMessageTime = 0

  Last message from control node.
- double currentTime = 0

  Current time holder.
- int EmergencyTime = 3

  Emergency time [s].
- bool FlagEmergency = 0

  Emergency flag.
- char Mode [1]

  Message mode description.
- int VelocityL

  Message left wheel velocity description.
- int VelocityR

  Message right wheel velocity description.
- unsigned int LoopCounter = 1

  Loop counter for debugging purpose.
- double RolloMax = ROLLO_SPEED_MAX

  Maximum speed of the Rollo.
- double RolloMin = ROLLO_SPEED_MIN

  Minimum speed of the Rollo.
- double RolloRange = RolloMax - RolloMin

  Range of speed of the Rollo.

6.2.1 Detailed Description

Communication between ROS and Rollo.

Author

Rabbia Asghar
Ernest Skrzypczyk

Date

18/2/16

Command prototype: **rosrun rollo rollo_comm _rate:=10 _ip:=192.168.0.120 _port:=900 _em:=3 _square:=0 _forwardtime:=25 _turntime:=6 _squarespeed:=0.4**

Parameters

rate	Command sending frequency of the node <!10 [Hz]>
ip	Internet protocl address of target robot <!192.168.0.120 [1]>
port	User datagram protocol taget connection port <!900 [1]>
em	Emergency time <!3 [s]>
square	Square test switch <!0 [1]>: • 0 – Off • 1 – Simple square test • 2 – Double square test • n – N-th order square test
forwardtime	Time for forward motion of robot <!25 [s]>
turntime	Time for turning the robot <!6 [s]>
squarespeed	Square test forward motion speed <!0.4 [1]>:

Provides basic communication structure between ROS holding nodes used for localization and Rollo.
Main aspects include:

- decode linear and angular velocities provided by control node
- translate and send message to Rollo
- publish decoded velocities
- square test or n-th order
- emergency procedure

Project github repository

See also

https://github.com/em-er-es/rollo/

6.2.2 Function Documentation

6.2.2.1 int decodeVelocities (double *x*, double *z*, char * *Message*, int & *VelocityL*, int & *VelocityR*)

Decode linear and angular velocities.

Velocities are decoded and stored as partial bytes of the UDP packet

Parameters

x	Linear velocity
z	Angular veolocity
&Message	UDP message to be send to target robot
VelocityL	Decoded velocity [%]
VelocityR	Decoded velocity [%]

Returns

0

Determine corresponding operation mode based on velocities

Since control node can provide abstract control values, an ideal case could be used for decoding velocities. This is discouraged, since using alternative control methods would probably have a realistic value set.

Linear velocity is approximately 0:

- Complete stop
- Right rotation
- Left rotation
- Lowest speeds for previous modes

Linear velocity is above tolerance threshold

Determine speeds based on the position of the "dial" z:

```
  |-a-|- - -*-b- - - - -|
 -1   z     0           1
```

Temporary velocity holder declaration

Eliminate problems with dividing through zero by adding a small number to variables

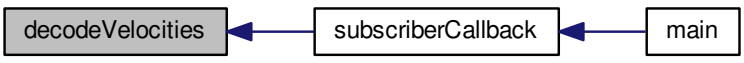
Calculate velocities according to relation expressed in linear and angular velocities ratio

Translate velocities for Rollo:

- Left wheel velocity - Second byte
- Right wheel velocity - Third byte
  - Temporary fix for errartic behaviour of Rollo

Determine forward or backward movement based on linear velocity

Here is the caller graph for this function:



6.2.2.2 void EstimateFeedbackVelocities (int *VelocityL*, int *VelocityR*, double & *LeftVelocityEstimate*, double & *RightVelocityEstimate*)

Assign left and right wheel velocity estimates for a given velocity command.

System odometry simulation in absence of encoder feedback using estimates from processed and analyzed data

Parameters

LeftVelocity↩ Estimate	Estimated velocity for left wheel determined from logs [rad/s]
RightVelocity↩ Estimate	Estimated velocity for right wheel determined from logs [rad/s]
VelocityL	Velocity command decoded from control node [%]
VelocityR	Velocity command decoded from control node [%]

Warning

Only velocities processed using logs are estimated.

Returns

0

- No movement
- Slowest movement speed – 6%
- Slow movement speed – 12%
- Lower medium movement speed – 19%
- Fastest movement speed in current configuration – 56%
- Combination of different movement speeds – L12% & R19%
- Combination of different movement speeds – L19% & R12%
- Combination of different movement speeds – L31% & R38%
- Combination of different movement speeds – L38% & R31%

Here is the caller graph for this function:

6.2.2.3 int main (int *argc*, char ∗∗ *argv*)

Node main.

Depending on specified parameters processes data from control node and Rollo and transmits them to appropriate targets or runs a square test of n-th order

Parameters

rate	Command sending frequency of the node <!10 [Hz]>
ip	Internet protocl address of target robot <!192.168.0.120 [1]>
port	User datagram protocol taget connection port <!900 [1]>
em	Emergency time <!3 [s]>
square	Square test switch <!0 [1]>: • 0 – Off • 1 – Simple square test • 2 – Double square test • n – N-th order square test
forwardtime	Time for forward motion of robot <!25 [s]>
turntime	Time for turning the robot <!6 [s]>
squarespeed	Square test forward motion speed <!0.4 [1]>:

Returns

0

Initalization

Initialize node

Initialize nodehandle

Initialie subscriber and define topic and message queue size

Publish velocities as [rpm]

Initialize node arguments using command line

Initialize node parameters from launch file or command line. Use a private node handle so that multiple instances of the node can be run simultaneously while using different parameters.

Node main parameters

Emergency parameters

Square test parameters Default values

Feedback velocities in rad/s to publish

Node frequency rate [Hz]

Initialize subscriber message type

Initialize publisher message type

Initialize variables for computing linear and angular velocity of the robot

Client initialization

Square test

Alternatively this square test could be in control node, however communication node is "closer" to Rollo

- Print information on current run
- Compose turn command
- Check square run variable and determine turning direction

For multiple runs the robot would go back and forth providing more reliable data on the actual error In ideal case even a high order square run would result in the robot being at the initial position with initial orientation

- Compose forward command
- Set initial time
- Initialize bytes sent variable for debugging
- Print square test parameters

Main square loop

1. Moving forward
    - Send command 3 times
    - Wait for the specified time to move forward
2. Turning
    - Send command 3 times
    - Wait for the specified time to turn

Main square loop end

- Update run finish time
- Print duration time
- Check square loop condition
    - Turn around
        - Send command 3 times
        - Wait for twice the specified time to turn around
    - Update square run counter and check for exit condition
- Stop
    - Send stop command 10 times
- Update square run counter and check for exit condition
- Use LoopCounter as square test run indicator

Square test end

Main loop

- Send control command to Rollo

Emergency procedure

- Check if emergency condition has been met:
- Print emergency message
- Conduct emergency stop

Send emergency message 10 times

- Hard condition emergency procedure
  Exit with an error code if hard condition is set by using negative values for emergency time

- Soft condition emergency procedure
  - Set emergency flag
  - Empty procedure sequence if emergency flag is raised
    - ROS spinOnce
    - Sleep before running loop again

Emergency procedure end

- Estimate feedback velocities
- Compose message to be published:
  - Assign timestamp
  - Assign estimated feedback velocities
- Publish message
- Print published message

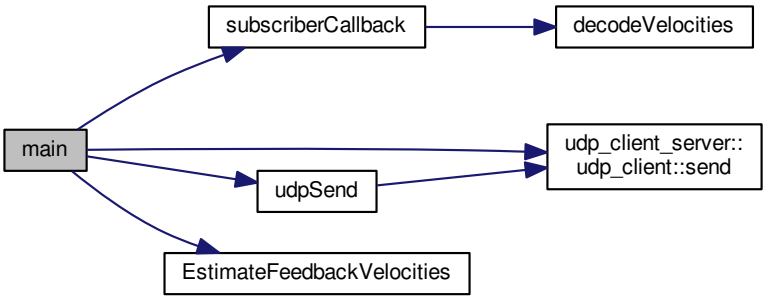
ROS spinOnce

Sleep before running loop again

Increase LoopCounter

Main loop end

Here is the call graph for this function:



6.2.2.4 void subscriberCallback (const geometry_msgs::Twist::ConstPtr & *msg*)

Subscriber callback from control node.

Read newest velocities from control node and translate them into UDP message. Update latest message time.

Parameters

msg	Message from control node containing linear and angular velocities

Returns

0

Update the UDP message

Update last message time

Reset emergency flag

Here is the call graph for this function:

Here is the caller graph for this function:

6.2.2.5 int udpSend (char *ip[16]*, int *port*, char * *Message*)

Send UDP packets.

Send provided message using included UDP library command `udp_client_server::udp_client.send()`

Parameters

&ip	IP address of the target robot
&port	UDP port of the target robot
&message	UDP message sent to robot

Returns

Bytes sent

Client initialization

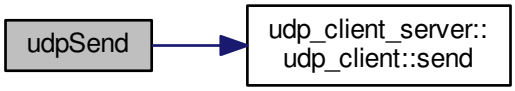
Send UDP message

Check if number of bytes sent is equal to bytes of composed message

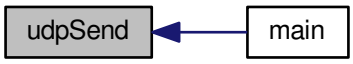
Error handling

Return bytes sent or error code

Here is the call graph for this function:



Here is the caller graph for this function:

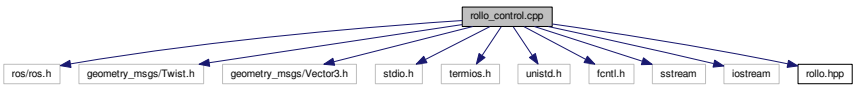


6.3 rollo_control.cpp File Reference

Convert input from keyboard and publish control commands for Rollo.

```
#include "ros/ros.h"
#include "geometry_msgs/Twist.h"
#include "geometry_msgs/Vector3.h"
#include <stdio.h>
#include <termios.h>
#include <unistd.h>
#include <fcntl.h>
#include <sstream>
#include <iostream>
#include "rollo.hpp"
```

Include dependency graph for rollo_control.cpp:



Functions

- int kbhit (void)

  Keyboard keystroke.
- void decodeKey (char character, double &Speed, double &Turn, double &LastTurn)

Decode key.

- int main (int argc, char **argv)

  Node main.

Variables

- char NodeName [20] = C2 CT CR

  Node name using console codes.
- char TopicCmdVel [64] = TOPIC_CTRL_CMD_VEL

  Topic for commands generated expressed in linear and angular velocity.
- const double LimitVelocityF = 1

  Limit velocity forward.
- const double LimitVelocityR = -1

  Limit velocity reverse.
- const double LimitTurnVelocityR = 0.50

  Limit turn velocity right.
- const double LimitTurnVelocityL = -0.60

  Limit turn velocity left.
- const double LKeysSteps = 0.1

  Left key set velocity step.
- const double RKeysLinearV = 0.4

  Right key set linear velocity step.
- const double RKeysAngularV = 1

  Right key set angular velocity step.

6.3.1 Detailed Description

Convert input from keyboard and publish control commands for Rollo.

Author

Rabbia Asghar
Ernest Skrzypczyk

Date

18/2/16

Command prototype: **rosrun rollo rollo_control _rate:=10**

Parameters

rate	Running frequency of the node <!10 [Hz]>

Robot control using following key sets

```
q    w    e          u    i    o
a    s    d    f/F   j    k    l
z    x    c          m    ,    .
```

Left key set:

- q/e : increase/decrease speeds 0.1 and -0.1
- w/s : increase/decrease only linear speed by 0.1

- a/d : increase/decrease only angular speed by 0.1
- z/c : increase/decrease speeds 0.1 and -0.1
- x : reset angular speed

Independent key set:

- f/F : full speed forwards/backwards

Right key set:

- u/o : increase/decrease set speeds for diagonal movement forwards
- i/, : increase/decrease set speeds for forward/backward movement
- j/l : increase/decrease set speeds for rotations
- m/. : increase/decrease set speeds for diagonal movement backwards
- k : stop

Global key set:

- $*$: stop
- <CTRL>-C : quit

Python script available online used as reference

See also

https://github.com/ros-teleop/teleop_twist_keyboard/blob/master/teleop_twist_keyboard.py

Project github repository

See also

https://github.com/em-er-es/rollo/

6.3.2 Function Documentation

6.3.2.1 void decodeKey (char *character*, double & *Speed*, double & *Turn*, double & *LastTurn*)

Decode key.

Compute linear and angular command velocities based on keyboard input. Key pressed character <key> as input argument.

Parameters

character	Character to be decoded
&Speed	Linear velocity
&Turn	Angular velocity

Returns

NULL

See also

https://github.com/ros-teleop/teleop_twist_keyboard/blob/master/teleop_twist_keyboard.py

Algorithm:

Switch character to decode:

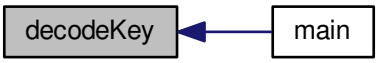
- Left key set control
- Full speed forward/backward
- Right key set control
- Default value

Velocity limits:

- Linear velocity limits
- Angular velocity limits

Print decoded velocities

Here is the caller graph for this function:



6.3.2.2 int kbhit (void)

Keyboard keystroke.

Check if a key is pressed on keyboard and return it.

Parameters

NONE	

Returns

1 if a key is pressed on keyboard, otherwise 0.

See also

https://github.com/sdipendra/ros-projects/blob/master/src/keyboard_non_blocking_input/src/keyboard_non_blocking_input_node.cpp

Here is the caller graph for this function:

6.3.2.3 int main (int *argc,* char ** *argv*)

Node main.

Initialize variables and nodehandle, read and translate input information into command messages.

Parameters

rate	Running frequency of the node <!10 [Hz]>

Publish to command velocity topic as specified in configuration header `rollo.hpp` according to format geometry_msgs::Twist

Returns

0

Algorithm structure

Initialization

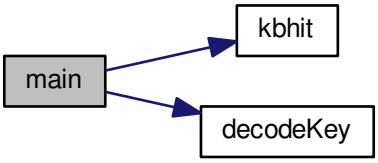
- Initialize nodehandle for publisher
- Publisher initialization with topic, message format and queue size definition
- Node arguments using command line
- Initialize node parameters from launch file or command line. Use a private node handle so that multiple instances of the node can be run simultaneously while using different parameters.
- Publishing rate [Hz]
- Publisher variables for conventional messages
- Initialize variables for computing linear and angular velocity of the robot
- Initialize character holder

Main loop

- Check if a key is pressed:
  - Read character
  - Decode key pressed
- Prepare message to publish linear and angular velocities
- Print message with velocities
- Publish message in Twist format
- ROS spinOnce
- Sleep to conform node frequency rate

Main loop end

Here is the call graph for this function:

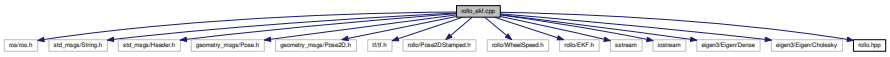


6.4 rollo_ekf.cpp File Reference

EKF implementation for localisation of a robot.

```
#include "ros/ros.h"
#include "std_msgs/String.h"
#include "std_msgs/Header.h"
#include "geometry_msgs/Pose.h"
#include "geometry_msgs/Pose2D.h"
#include "tf/tf.h"
#include "rollo/Pose2DStamped.h"
#include "rollo/WheelSpeed.h"
#include "rollo/EKF.h"
#include <sstream>
#include <iostream>
#include <eigen3/Eigen/Dense>
#include <eigen3/Eigen/Cholesky>
#include "rollo.hpp"
```

Include dependency graph for rollo_ekf.cpp:



Functions

- void subscriberCallbackMeasurement (const rollo::Pose2DStamped msg)

  SubscriberCallbackMeasurement.
- void subscriberCallbackControlInput (const rollo::WheelSpeed msg)
- rollo::WheelSpeed interpolateOdometry (rollo::WheelSpeed OdometryPrev, rollo::WheelSpeed Odometry↩ Current, double EKFfilterTimeSecs)

  Linear interpolation of values from odometry.
- rollo::Pose2DStamped interpolateMeasurement (rollo::Pose2DStamped zPrev, rollo::Pose2DStamped z↩ Current, double EKFfilterTimeSecs)

  Linear interpolation of values from measurement (motion capture)
- Eigen::Vector3d FSTATE (Eigen::Vector3d x_pp, Eigen::Vector2d u)

  Nonlinear state equation function f(x_k-1, u_k-1)

- Eigen::Matrix3d JacobianFSTATE (Eigen::Vector3d x_pp, Eigen::Vector2d u)

  Linearization of f(x_k-1, u_k-1)
- Eigen::Vector3d HMEAS (Eigen::Vector3d x_cp)

  Measurement function h(x_k)
- int main (int argc, char ∗∗argv)

  Node main.

Variables

- rollo::Pose2DStamped zPose2DStamped

  Measurement message includes Pose2D along with timestamp.
- double zTimeSecs = 0

  Initialize variable that save timestamps from both measurement subscriber in double (float64 in message format).
- rollo::WheelSpeed Odometry

  Odometry message includes timestamp and angular velocities for left and right wheel.
- double OdometryTimeSecs = 0

  Initialize variable that save timestamps from odometry subscriber in double (float64 in message format).
- char NodeName [20] = C4 KF CR

  Node name using console codes.
- char TopicEKF [64] = TOPIC_EKF

  Topic for extended Kalman filter results with all three estimated states and covariance matrix, stamped.
- char TopicWheelSpeed [64] = TOPIC_COMM_WS

  Topic for wheel speed containing the actual speed of wheel, preferably extracted from encoders or if not available by using a lookup table.
- char TopicPose2DStamped [64] = TOPIC_PREP_P2DT

  Topic for position and orientation stamped from preprocessor node.

6.4.1 Detailed Description

EKF implementation for localisation of a robot.

Author

Rabbia Asghar
Ernest Skrzypczyk

Date

20/2/16

Command prototype: **rosrun rollo rollo_ekf _rate:=10**

Parameters

rate	Sampling frequency of the node <!10 [Hz]>

Based on control input from communication node in form of control commands and measurement from preprocessor node, extended Kalman filter implementation estimates of states for localization and publishes estimated states with covariance. Additional information in form of odometry based state estimate are also published for easier analysis of the filter results. Kalman filter equations were first implemented in MATLAB, then translated into C++, compared and verified with test values.

Localization of the robot consists of 3 states:

- Position (x, y)

- Orientation (Theta)

For initial state estimate the node uses robots position and orientation taken from Pose2D message from preprocessor node before running EKF iterations. Initial state covariance matrix is taken as an identity matrix.

Timing for EKF update is inspired from Robot Pose EKF (robot/pose/ekf) package available for ROS:
Timings and data at those specific time instants are synchronized in such a manner, that the latest measurements with newer timestamps are interpolated to one and the same timestamp, when all necessary data is available. This allows for a relative comparison of available data, even though an additional error is introduced through interpolating.

See also

http://wiki.ros.org/robot_pose_ekf

Project github repository

See also

https://github.com/em-er-es/rollo/

6.4.2 Function Documentation

6.4.2.1 Eigen::Vector3d FSTATE (Eigen::Vector3d *x_pp*, Eigen::Vector2d *u*)

Nonlinear state equation function f(x_k-1, u_k-1)

This is part of time update (or prediction update) of EKF. Given a priori state estimate x_k-1|k-1 and u_k-1, it computes predicted value for state x_k|k-1.

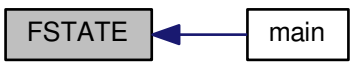
Parameters

x_pp	contains a priori state estimate, x_k-1|k-1
u	is control input vector, calculated from odometry. It consists of 2 elements, delta S and delta theta

Returns

Eigen::Vector3d, state prediction x_k|k-1

Here is the caller graph for this function:



6.4.2.2 Eigen::Vector3d HMEAS (Eigen::Vector3d *x_cp*)

Measurement function h(x_k)

This computes estimated measurement vector based on the latest state estimate.

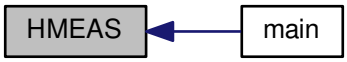
Parameters

x_cp	contains state prediction x_k|k-1 computed in time update of EKF

Returns

Eigen::Vector3d, contains estimated measurement vector

Here is the caller graph for this function:



6.4.2.3 rollo::Pose2DStamped interpolateMeasurement (rollo::Pose2DStamped *zPrev,* rollo::Pose2DStamped *zCurrent,* double *EKFfilterTimeSecs*)

Linear interpolation of values from measurement (motion capture)

This function performs linear interpolation of robot pose2D for a given time instant. The time for which the robot pose2D are computed is defined by EKFfilterTimeSecs.

Parameters

zPrev	contains robot pose2D and time stamp read at previous instant when EKF was updated.
zCurrent	contains robot pose2D and time stamp read currently.
EKFfilterTime↩ Secs	is the time instant for which the EKF update need to be performed and robot pose2D need to be computed.

Returns

rollo::Pose2DStamped, contains robot pose2D computed for time instant given by EKFfilterTimeSecs using linear interpolation.

Here is the caller graph for this function:

6.4.2.4 rollo::WheelSpeed interpolateOdometry (rollo::WheelSpeed *OdometryPrev,* rollo::WheelSpeed *OdometryCurrent,* double *EKFfilterTimeSecs*)

Linear interpolation of values from odometry.

This function performs linear interpolation of right and left wheel speed for a given time instant. The time for which the odometry values are computed is defined by `EKFfilterTimeSecs`.

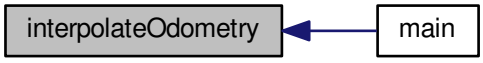
Parameters

OdometryPrev	contains left and right wheel speed and time stamp read at previous instant when EKF was updated.
Odometry↩ Current	contains left and right wheel speed and time stamp read currently.
EKFfilterTime↩ Secs	is the time instant for which the EKF update need to be performed and odometry values need to be computed.

Returns

rollo::WheelSpeed, contains left and right wheel speed [rad/s] computed for time instant given by `EK↩ FfilterTimeSecs` using linear interpolation.

Here is the caller graph for this function:



6.4.2.5 Eigen::Matrix3d JacobianFSTATE (Eigen::Vector3d *x_pp,* Eigen::Vector2d *u*)

Linearization of f(x_k-1, u_k-1)

This is part of time update(or prediction update) of EKF. This computes Jacobian matrix by taking the partial derivatives of f(x_k-1,u_k-1) with respect to x, evaluated at the last state estimate x_k-1|k-1.

Parameters

x_pp	contains a priori state estimate x_k-1|k-1
u	is control input vector, calculated from odometry. It consists of 2 elements, delta S and delta theta

Returns

Eigen::Matrix3d is the Jacobian matrix

Here is the caller graph for this function:

6.4.2.6 int main (int *argc,* char ** *argv*)

Node main.

Initialize node, nodehandle, subscribe to messages from preprocessor and communication nodes and publish estimated state of the robot.

Parameters

rate	Sampling frequency of the node <!10 [Hz]>

Initializes Extended Kalman Filter revelant variables. As a part of initializing, function waits for one message from each subscriber and save timestamps for the first iteration of EKF. State estimate, x_(0|-1) is initialized as the first measurement read from the preprocessor node. Covariance of state estimate, E(0, -1) is initialized as identity matrix.

Run EKF in loop, update estimates. Await new sensor data, determine time step for EKF update and perform necessary interpolation.

Publishes newest estimates of state variables, covariance matrix and timestamp.

Returns

0

Initialize

- Initialize node
- Initialize nodehandle for subscribers and publisher
- Initialize subscribers:
  - Initialize subscriber for measurement data
  - Initialize subscriber for actual speed of wheels, preferably extracted from encoders or if not available by using a lookup table
- Initialize publisher and define topic and message queue size for the publisher
- Initialize node arguments using command line
- Initialize node parameters from launch file or command line Use a private node handle so that multiple instances of the node can be run simultaneously while using different parameters.
- Publishing rate [Hz]
- Loop condition and counter variable
- Initialize variables involved in computation of EKF:
  - Define number of states
  - Initialize process noise covariance matrix
  - Initialize measurement noise covariance matrix
  - Initialize vector for control input u and variables involved in its computation
  - Initialize state estimate vector a priori
  - Initialize Jacobian matrix with the partial derivatives of h(x_k) with respect to x, identity for given system
  - Initialize E_pp: a priori estimated state covariance, E_k-1|k-1 (p refers to k-1)
  - Initialize variables involved in the prediction update of EKF
  - Initialize variables involved in the innovation update of EKF
  - Initialize state estimate vector and state covariance matrix a posteriori
  - Variables involved in odometry update alone
  - Variables for determining EKF time step
  - Variables involved in interpolation of odometry and measurement data
  - Control input and measurement variables used in EKF update

EKF Initialization loop

- Check if data is available from measurement (motion capture)
  - Initialize `prevzPose2DStamped`, `prevMeasurementSecs` and `PreviousEKFfilterTimeSecs`
- Check if data is available from odometry (control input)
  - Initialize `prevOdometry`, `prevOdometrySecs` and `PreviousEKFfilterTimeSecs`
- Check if new data has been read from both measurement (motion capture) and odometry (control input)
  - Initialize initial state estimate `x_pp`

Initialization loop end

Main loop

- Check if new data is available from measurement (motion capture) and odometry (control input)
  - If new data is available and measurement data is for timestamp later than odometry's, perform interpolation for measurement
- Update timestamp
- Interpolate measurements
- Update measurement and control input for EKF update
- If new data is available and odometry data is for time stamp later than measurement's, perform interpolation for odometry
  - Update timestamp
  - Interpolate odometry
  - Update measurement and control input for EKF update
- Update `prevOdometry` and `prevzPose2DStamped` for next loop

EKF update

Perform EKF update if all sensor data is available:

- Determine time step for EKF update
- Update `PreviousEKFfilterTimeSecs` for the next loop
- Determine control input u from left and right wheel speed for EKF update
- Prediction update
- Update state prediction based on a priori state estimate and control input
- Update predicted state estimate covariance matrix based on a priori state and control input
- Innovation update
- Estimate measurement based on a priori state estimate
- Perform Cholesky decomposition: Instead of standard equations for EKF, use Cholesky factorization for a stable covariance matrix
- Compute a posteriori state estimate x_k|k and a posteriori state covariance matrix E_k|k
- Update a priori state estimate x_pp and a priori state covariance matrix E_pp for next loop

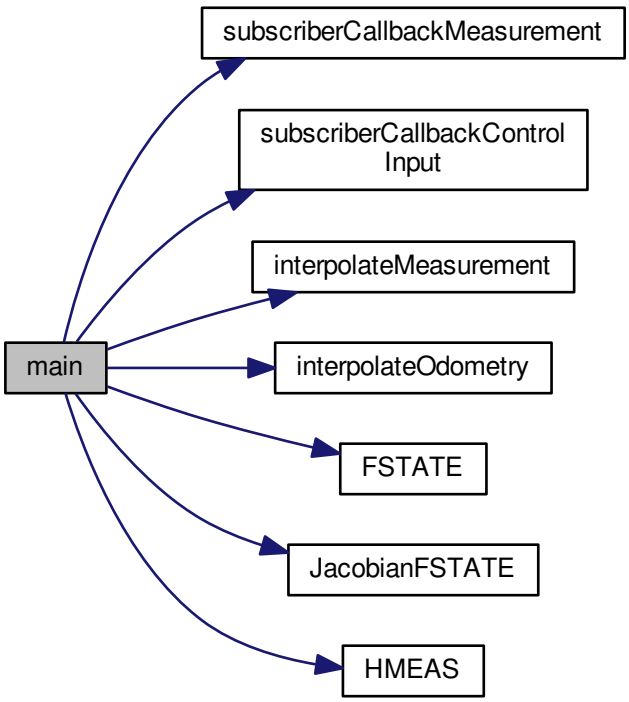
- Determine odometry update without extended Kalman filter
- Prepare data for publishing
- Pose2D EKF
- Covariance
- Pose2D odometry
- Publish

EKF Update end

- Synchronize to rate

Main loop end

Here is the call graph for this function:



6.4.2.7 void subscriberCallbackControlInput (const rollo::WheelSpeed *msg*)

Read new message

Here is the caller graph for this function:

6.4.2.8 void subscriberCallbackMeasurement (const rollo::Pose2DStamped *msg*)

SubscriberCallbackMeasurement.

Subscribe to the topic '/Rollo/preprocessor/pose2dstamped' of the preprocessor node. Read filtered position and orientation of the robot and timestamp. Update global variables `zPose2DStamped` and `zTimeSecs` for use in EKF update.

Parameters

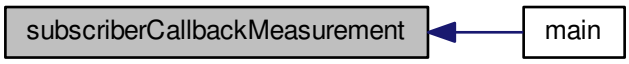
msg	- custom defined message (preprocessor node).

Returns

NULL

Read new message

Here is the caller graph for this function:

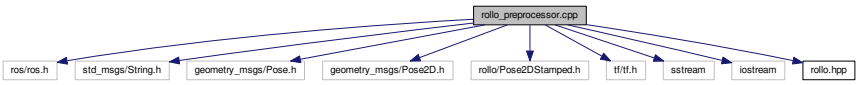


6.5 rollo_preprocessor.cpp File Reference

Preprocessor for Rollo measurement using Mocap OptiTrack motion capture data.

```
#include "ros/ros.h"
#include "std_msgs/String.h"
#include "geometry_msgs/Pose.h"
#include "geometry_msgs/Pose2D.h"
#include "rollo/Pose2DStamped.h"
#include "tf/tf.h"
#include <sstream>
#include <iostream>
#include "rollo.hpp"
```

Include dependency graph for rollo_preprocessor.cpp:



Functions

- void subscriberCallback (const geometry_msgs::Pose2D::ConstPtr &msg)

  Subscriber callback.
- int main (int argc, char **argv)

  Node main.

Variables

- char NodeName [20] = C1 PP CR

  Node name using console codes.
- double x

  Absolute coordinates.
- double **y**
- double **theta**
- double x_mm

  Absolute coordinates in various units.
- double **y_mm**
- double **theta_deg**
- char TopicMotionCapture [64] = TOPIC_PREP_MC

  Topic for motion capture data.
- char TopicPose2DStamped [64] = TOPIC_PREP_P2DT

  Topic for position and orientation, stamped.

6.5.1 Detailed Description

Preprocessor for Rollo measurement using Mocap OptiTrack motion capture data.

Author

Rabbia Asghar
Ernest Skrzypczyk

Date

16/2/16

Command prototype: **rosrun rollo rollo_preprocessor _rate:=25 _samplesize:=4 _sampling:=0**

Parameters

rate	Sampling frequency of the node <!25 [Hz]>
samplesize	Number of elements that are averaged/subsampled <!4 [1]>
sampling	Selects if the raw data should be subsampled after a certain delay or averaged over a certain period <!0 [1]> • sampling 0 sets subsampling • sampling !0 sets averaging

Filter the raw data from optitrack motion capture system and publish it along with time stamp for modeling of odometry and the measurement in Kalman filter.

Project github repository

See also

https://github.com/em-er-es/rollo/

6.5.2 Function Documentation

6.5.2.1 int main (int *argc*, char ** *argv*)

Node main.

Initialize variables, nodehandle, subscribe to motion capture data from mocap_optitrack node and publish position and orientation after processing with time stamp. The position and orientation are published along with timestamp in custom defined message format rollo::Pose2DStamped.

Parameters

rate	Sampling frequency of the node <!25 [Hz]>
samplesize	Number of elements that are averaged/subsampled <!4 [1]>
sampling	Selects if the raw data should be subsampled after a certain delay or averaged over a certain period <!0 [1]> • sampling 0 sets subsampling • sampling !0 sets averaging

Returns

0

Initialization

Name of the preprocessor node

- Nodehandle for subscriber and publisher
- Subscriber
- Publisher initialization with topic, message format and queue size definition
- Node arguments using command line

Sampling is either done using subsampling (0) or simple averaging (1)

- Initialize node parameters from launch file or command line. Use a private node handle so that multiple instances of the node can be run simultaneously while using different parameters.

- Publishing rate [Hz]
- Publisher variables for conventional messages
- Message type
- Publisher variables for processing
- Initialize variable to publish message
- Loop counter holder

Main loop

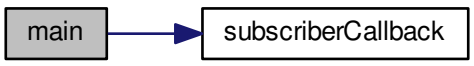
- Prepare data for publishing
- Publish
- Reset variables
- For subsampling sleep for time defined by rate and then read the states from the subscriberCallback() without usleep() delay

For averaging sleep for time defined by rate before reading states from the subscriberCallback()

- Increase loop counter

Main loop end

Here is the call graph for this function:



6.5.2.2 void subscriberCallback (const geometry_msgs::Pose2D::ConstPtr & *msg*)

Subscriber callback.

Subscribe to motion capture data from mocap_optitrack node and read position and orientation from Optitrack node.

Parameters

msg	Message generated by mocap_optitrack node in format: • Position x [m] • Position y [m] • Orientation [rad]

See also

https://github.com/ros-drivers/mocap_optitrack

Returns

NULL

Acquisition:

- Raw x coordinate [m]
- Raw y coordinate [m]
- Raw theta [rad]

Conversion into degrees in the range 0 to 360 degress

Print message with acquired data

Here is the caller graph for this function:

6.6 rollo_visualization.py File Reference

Visualize motion capture data and EKF estimates.

Classes

- class rollo_visualization.ProcessPlotter
- class rollo_visualization.MultiProcessPlot

Functions

- def **rollo_visualization.subscriberCallbackMeasurement** (msg)
  Subscriber callback for measurement data.
- def **rollo_visualization.subscriberCallbackEKF** (msg)
  Subscriber callback for EKF data.
- def **rollo_visualization.initAnimation** ()
  Initialization function for animation.
- def **rollo_visualization.animatePlot** (i)
  Animation callback function.
- def **rollo_visualization.initPlot** (object)
  Initilize plot.
- def **rollo_visualization.generatePlot** (initcond)
  Generate and update plot.
- def **rollo_visualization.RolloVisualization** ()
  Node class function.

Variables

- string **rollo_visualization.NodeName** = "\033[38;5;160mVIS \033[0m"

  Node name using console codes.
- int **rollo_visualization.rate** = 25

  Visualize rate [Hz] == [fps].
- int **rollo_visualization.plotRefreshPeriod** = 100

  Plot refresh period [1].
- int **rollo_visualization.LoopCounter** = 1

  Loop counter.
- int **rollo_visualization.markerScale** = 20

  Marker scale.
- int **rollo_visualization.axl** = 4

  Maximal coordinates - symmetrical
  Negative value used to mirror the current calibration setup of motion capture system and keep sanity with adjustments to the plot
  Positive value used to represent the current calibration setup of motion capture system and keep sanity with adjustments to the plot.
- **rollo_visualization.axlx** = axl

  X axis limit.
- **rollo_visualization.axly** = axl

  Y axis limit.
- **rollo_visualization.flagSubscriber1** = False

  Global message flag 1 – Motion capture data from preprocessor.
- **rollo_visualization.flagSubscriber2** = False

  Global message flag 2 – Extended Kalman filter estimates and odometry modeled data from EKF node.

6.6.1 Detailed Description

Visualize motion capture data and EKF estimates.

Author

Rabbia Asghar
Ernest Skrzypczyk

Date

25/2/16

Command prototype: **rosrun rollo rollo_visualization _rate:=25 _plotrefreshperiod:=100**

Parameters

rate	Running frequency of the node <!25 [Hz]>
plotrefreshperiod	Plot refresh period <!100 [1]>
ms	Marker scale reference value <20 [1]>
saveim	Save path for generated images <!.>
savevid	Save path for generated animation video <!.>
imtype	Type of saved images <!png>

imformat	Format of saved images (dim_x x dim_y) <!512>
duration	Duration of visualization <!0>

Warning

Not all parameters and functions are currently processed

Project github repository

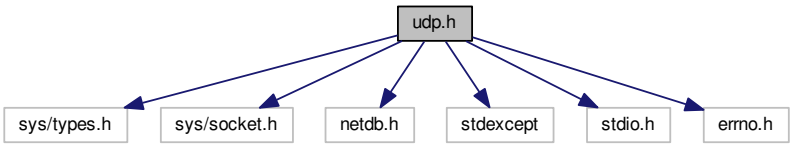
See also

https://github.com/em-er-es/rollo/

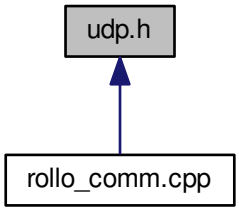
6.7 udp.h File Reference

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <stdexcept>
#include <stdio.h>
#include <errno.h>
```

Include dependency graph for udp.h:



This graph shows which files directly or indirectly include this file:



Classes

- class udp_client_server::udp_client_server_runtime_error
- class udp_client_server::udp_client
- class udp_client_server::udp_server

Index