Rollo - Localization of a humanoid robot

1.0

Generated by Doxygen 1.8.10

Thu Feb 25 2016 04:21:31

Contents

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

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

Chapter 4

Class Documentation

4.1 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.

4.1.1 Constructor & Destructor Documentation

4.1.1.1 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.

4.1.1.2 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.

4.1.2 Member Function Documentation

4.1.2.1 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.

4.1.2.2 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.

4.1.2.3 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.

4.1.2.4 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

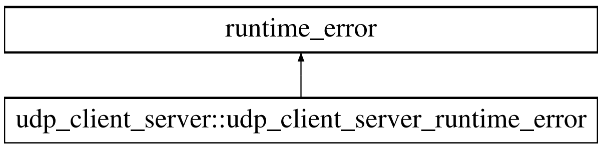
-1 if an error occurs, otherwise the number of bytes sent. errno is set accordingly on error.

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

- /home/emeres/.hdd/Documents/Studia/Università degli studi di Genova/DIBRIS/Tutti i corsi/86805 - SOFT↩ WARE ARCHITECTURES FOR ROBOTICS/AS01/rollo/src/udp.h
- /home/emeres/.hdd/Documents/Studia/Università degli studi di Genova/DIBRIS/Tutti i corsi/86805 - SOFT↩ WARE ARCHITECTURES FOR ROBOTICS/AS01/rollo/src/udp.cpp

4.2 udp_client_server::udp_client_server_runtime_error Class Reference

Inheritance 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:

- /home/emeres/.hdd/Documents/Studia/Università degli studi di Genova/DIBRIS/Tutti i corsi/86805 - SOFT↩ WARE ARCHITECTURES FOR ROBOTICS/AS01/rollo/src/udp.h

4.3 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.

4.3.1 Constructor & Destructor Documentation

4.3.1.1 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.

4.3.1.2 udp_client_server::udp_server::∼udp_server ( )

Clean up the UDP server.

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

4.3.2 Member Function Documentation

4.3.2.1 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.

4.3.2.2 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.

4.3.2.3 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.

4.3.2.4 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.

4.3.2.5 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:

- /home/emeres/.hdd/Documents/Studia/Università degli studi di Genova/DIBRIS/Tutti i corsi/86805 - SOFT↩ WARE ARCHITECTURES FOR ROBOTICS/AS01/rollo/src/udp.h
- /home/emeres/.hdd/Documents/Studia/Università degli studi di Genova/DIBRIS/Tutti i corsi/86805 - SOFT↩ WARE ARCHITECTURES FOR ROBOTICS/AS01/rollo/src/udp.cpp

Chapter 5

File Documentation

5.1 /home/emeres/.hdd/Documents/Studia/Università degli studi di Genova/DIBRIS/Tutti i corsi/86805 - SOFTWARE ARCHITECTURES FOR ROBOTICS/AS01/rollo/src/rollo_↩ comm.cpp File Reference

Communication between nodes 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"
```

Functions

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

  Range of speed of the Rollo.
- void subscriberCallback (const geometry_msgs::Twist::ConstPtr &msg)

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

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

  Main function.

Variables

- char NodeName [20] = C3 CM CR

  Global variables updated in the SubscriberCallback function, processed and used to send commands to the specified Ip adress at the UDP port.
- char TopicWheelSpeed [64] = TOPIC_COMM_WS

  Topics.
- char **TopicCmdVel** [64] = TOPIC_CTRL_CMD_VEL
- char ip [16] = "192.168.0.120"

  Ip address.

- int port = 900

  UDP port.
- double tol = 0.01

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

  Velocities for both wheels.
- 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.
- char Mode [2]

  Message mode description.
- int **VelocityL**
- int VelocityR

  Message velocities description.
- unsigned int **loopcounter** = 1
- double RolloMax = ROLLO_SPEED_MAX

  Maximum speed of the Rollo.
- double RolloMin = ROLLO_SPEED_MIN

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

5.1.1 Detailed Description

Communication between nodes and Rollo.

Author

Rabbia Asghar, Ernest Skrzypczyk

Date

18/2/16

See also

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

5.1.2 Function Documentation

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

Range of speed of the Rollo.

Decode linear and angular velocities

Velocities are decoded and stored as partial bytes of the UDP packet Parameters declared: `x`, `z`, `&Message`. Parameters declared: `x`, `z`, `&MessageB1`, `&MessageB2`, `&MessageB3`.

Returns

0

Determine corresponding operation mode based on velocities

Complete stop

Right rotation

Left rotation

Lowest speeds for previous modes

Determine speeds based on the position of the "dial" z
|-a-|- - -*-b- - - - -|
-1 z 0 1

< Temporary velocity holder

< Eliminate problems with dividing through zero by adding

Temporary fix for errartic behaviour of Rollo

Determine forward or backward movement

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

Main function.

Returns

0

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.

Square test parameters

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 should be in control node, however communication node is "closer" to Rollo

Compose turn command

Compose forward command

Bytes sent, useful for debugging

Moving forward

Send command 3 times

Turn

Send command 3 times

Main loop

Send control command to Rollo

Publish encoder readings

Compose message

Publish message

ROS spinOnce

Sleep before running loop again

Main loop end

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

Subscriber callback.

Reads newest velocities from control node and decodes them into UDP message

Returns

0

See also

DOCURLCallback function for subscriber

Update the UDP message

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

Send UDP packets.

Parameters declared by reference: `&ip`, `&port`, `&message`.

Returns

Bytes sent

See also

DOCURL

Client initialization

5.1.3 Variable Documentation

5.1.3.1 char NodeName[20] = C3 CM CR

Global variables updated in the SubscriberCallback function, processed and used to send commands to the specified Ip adress at the UDP port.

Node name using console codes

5.2 /home/emeres/.hdd/Documents/Studia/Università degli studi di Genova/DIBRIS/Tutti i corsi/86805 - SOFTWARE ARCHITECTURES FOR ROBOTICS/AS01/rollo/src/rollo_↩ control.cpp File Reference

Takes input from keyboard and publishes commands to control 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"
```

Functions

- int kbhit (void)

  kbhit function
- void decodeKey (char character, double &Speed, double &Turn)

  Decode key.
- int main (int argc, char **argv)

  Main.

Variables

- char NodeName [20] = C2 CT CR

  Global variables.
- char **TopicCmdVel** [64] = TOPIC_CTRL_CMD_VEL
- double **VelocityFwd_limit** = 1
- double **VelocityRev_limit** = -1
- double **LKeysSteps** = 0.1
- double **RKeysLinearV** = 0.4
- double **RKeysAngularV** = 1

5.2.1 Detailed Description

Takes input from keyboard and publishes commands to control Rollo.

Author

Rabbia Asghar

Date

18/2/16

Robot control using following keys

Moving around: u i o j k l

m , .

q/z : increase/decrease speed by 0.1 w/x : increase/decrease only linear speed by 0.1

e/c : increase/decrease only angular speed by 0.1

Other keys : 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

5.2.2 Function Documentation

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

Decode key.

Computes command linear and angular velocities depending on the keyboard key pressed. The function takes keyboard key pressed character `as` input argument. Parameters declared by reference: `&Speed`, and `&Turn`.

Returns

NULL

See also

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

Turn limits

5.2.2.2 int kbhit (void)

kbhit function

Checks if a key is pressed on keyboard and returns it.

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

5.2.2.3 int main (int *argc,* char $**$ *argv*)

Main.

Initializes variables, nodehandle, reads and translated input information into command messages. Accepts 1 argument from command line: rate. Default rate is 10 [Hz]. The commands are published to topic /Rollo/cmd_vel in configuration in format geometry_msgs::Twist

Returns

0

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

Main loop

Read if any keyboard key is pressed

Read character

Decode key pressed

Prepare message to publish linear and angular velocity

Publish message in Twist format

ROS spinOnce

Sleep before running loop again

Main loop end

5.3 /home/emeres/.hdd/Documents/Studia/Università degli studi di Genova/DIBRIS/Tutti i corsi/86805 - SOFTWARE ARCHITECTURES FOR ROBOTICS/AS01/rollo/src/rollo_↩ ekf.cpp File Reference

EKF implementation for localisation of the 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 "rollo.hpp"
```

Functions

- void subscriberCallbackMeasurement (const rollo::Pose2DStamped msg)

  SubscriberCallbackMeasurement.

- void subscriberCallbackControlInput (const rollo::WheelSpeed msg)
- rollo::WheelSpeed interpolateOdometry (rollo::WheelSpeed Odometryold, rollo::WheelSpeed Odometrynew, double EKFfilterTimeSecs)

  Linear interpolation of values from odometry.
- rollo::Pose2DStamped interpolateMeasurement (rollo::Pose2DStamped zOld, rollo::Pose2DStamped zNew, double EKFfilterTimeSecs)

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

  FSTATE nonlinear state equations, f(x_k-1, u_k-1)
- Eigen::Matrix3d JacobianFSTATE (Eigen::Vector3d x_pp, Eigen::Vector2d u)

  JacobianFSTATE.
- Eigen::Vector3d HMEAS (Eigen::Vector3d x_cp)

  HMEAS measurement equation, h(x_k)
- int main (int argc, char ∗∗argv)

  Main function.

Variables

- rollo::Pose2DStamped zPose2DStamped

  Global variables updated in the SubscriberCallback functions.
- double **zTimeSecs** = 0
- rollo::WheelSpeed **Odometry**
- double **OdometryTimeSecs** = 0
- char **NodeName** [20] = C1 KF CR
- char TopicEKF [64] = TOPIC_EKF

  Topics.
- char **TopicWheelSpeed** [64] = TOPIC_COMM_WS
- char **TopicPose2DStamped** [64] = TOPIC_PREP_P2DT

5.3.1 Detailed Description

EKF implementation for localisation of the robot.

Author

Rabbia Asghar, Ernest Skrzypczyk

Date

20/2/16

Takes control input from communication node and measurement from preprocessor node, implements Extended Kalman Filter for the estimation of states and publishes estimated state. For localisation of the robot there are 3 states: position (x, y), orientation (Theta).

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, where 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

Kalman filter equations were first simulated in MATLAB and then written in C++ and compared and verified by results in MATLAB.

Command: rosrun rollo rollo_ekf _rate:=1

- rate is the sampling frequency of the node (default: 1)

  See also

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

5.3.2 Function Documentation

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

FSTATE nonlinear state equations, 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.

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

HMEAS measurement equation, 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.

5.3.2.3 rollo::Pose2DStamped interpolateMeasurement (rollo::Pose2DStamped *zOld*, rollo::Pose2DStamped *zNew*, 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

zOld	contains robot pose2D and timestamp read at previous instant when EKF was updated.
zNew	contains robot pose2D and timestamp 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.

5.3.2.4 rollo::WheelSpeed interpolateOdometry (rollo::WheelSpeed *Odometryold,* rollo::WheelSpeed *Odometrynew,* 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

Odometryold	contains left and right wheel speed and timestamp read at previous instant when EKF was updated.
Odometrynew	contains left and right wheel speed and timestamp 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.

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

JacobianFSTATE.

This computes Jacobian matrix by taking the partial derivatives of f(x_k-1,u_k-1) w.r.t x

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

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

Main function.

Initialize node, nodehandle, subsrcribe to messages from preprocessor and communication nodes and publish estimated state of the robot. Arguments from command line: rate.

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 node

Initialize nodehandle for subscribers and publisher

Initialize subscribers

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 in units of Hz

Initialize variables involved in computation of EKF Define number of states

Initialize noise covariances and matrices

Initialize vector for control input u and variables involved in its computation

Initialize state estimate vector "a priori" and measurement vector

Initialize Jacobian matrix with the partial derivatives of h(x_k) w.r.t x, identity for provided 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 for time

Initialize measurement vector with timestamp and odometry data with timestamp from subscriber messages Initialize state estimate using measurement vector reading

Initialization loop

Initialization loop end

Main loop

Check if new data is available from measurement (motion capture) and odometry (control input)

Determine time step for EKF update and perform interpolation for the sensor data not available at respective time step

Update timestamp

Interpolate measurements

Update state

Update timestamp

Interpolate for measurement

Update variables involved in EKF update

Update `prevOdometry` and `prevzPose2DStamped` for next loop

Perform EKF update if all sensor data is available

Determine dt

Update PreviousEKFfilterTimeSecs for the next loop

Determine control input u from nL and nR

Prediction update Nonlinear update and linearization at current state

Partial covariance update

Innovation update Nonlinear measurement and linearization

Update E_pp an x_pp for next loop for next loop

Prepare data for publishing

Pose2D

Covariance

Publish

Synchronize to rate

Main loop end

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

Read new message

5.3.2.8 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

5.3.3 Variable Documentation

5.3.3.1 rollo::Pose2DStamped zPose2DStamped

Global variables updated in the SubscriberCallback functions.

Initialize custom defined messages for meaurement and odometry. Measurement message includes Pose2D along with timestamp. Odometry message includes timestamp and angular velocities for left and right wheel. Initialize variables that save timestamps from both measurement and odometry subscribers in double (float64 in message format).

Index