Latest Features in Robotics System Toolbox

September 2016

R2016b

Robotic Manipulator Algorithms

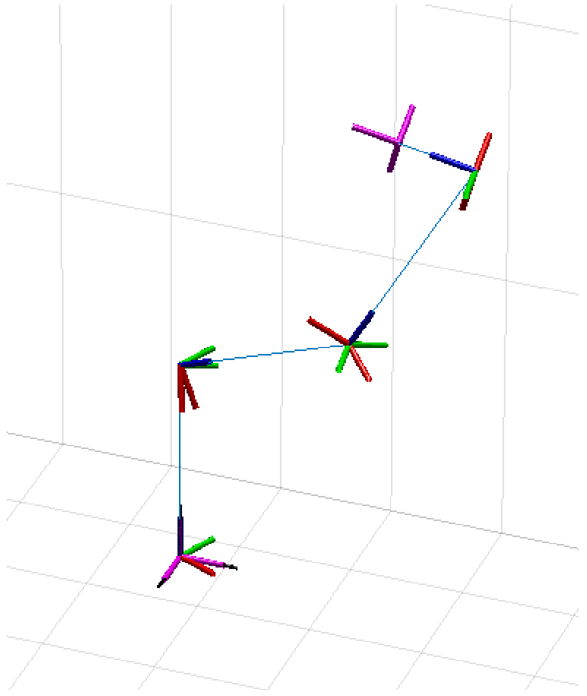
Represent robot manipulators using a rigid body tree

- Build kinematic chains or trees using rigid bodies to represent physical robots with the `robotics.RigidBodyTree` class

- Add or modify bodies on a structure, specify joint limits, and replace bodies or joints

- Support for revolute, prismatic, and fixed joints

- Simple visualization of body frames



```
>> showdetails(lbr)
--------------------
Robot: (9 bodies)

 Idx     Body Name            Joint Name            Joint Type    Parent Name(Idx)   Children Name(s)
 ---     ---------            ----------            ----------    ----------------   ----------------
   1        link_1              joint_a1             revolute         base_link(0)   link_2(2)
   2        link_2              joint_a2             revolute            link_1(1)   link_3(3)
   3        link_3              joint_a3             revolute            link_2(2)   link_4(4)
   4        link_4              joint_a4             revolute            link_3(3)   link_5(5)
   5        link_5              joint_a5             revolute            link_4(4)   link_6(6)
   6        link_6              joint_a6             revolute            link_5(5)   link_7(7)
   7        link_7              joint_a7             revolute            link_6(6)   tool0(8)
   8         tool0        joint_a7_tool0                fixed            link_7(7)
   9          base        base_link_base                fixed         base_link(0)
--------------------
```

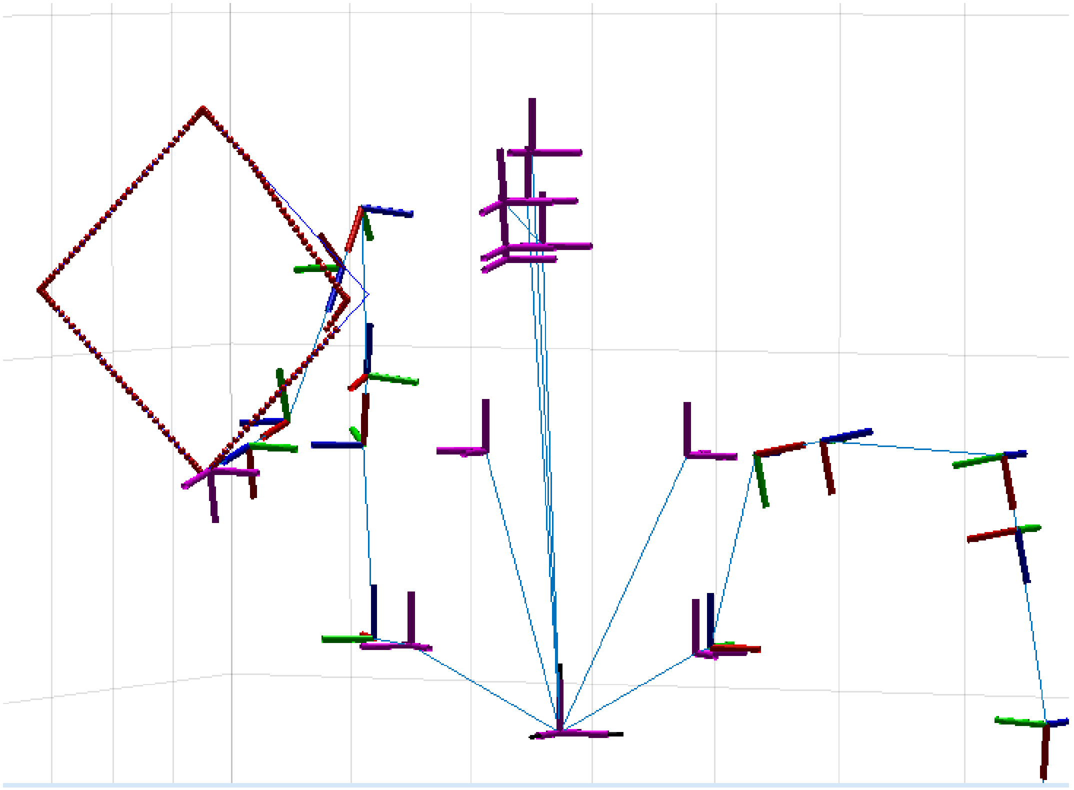
```
» load exampleRobots
```

Robotic Manipulator Algorithms

R2016b

Calculate forward and inverse kinematics for rigid body trees

- Use forward kinematics to get transformations between two body frames
- Compute geometric Jacobians for specified end effectors
- Use `robotics.InverseKinematics` class to calculate corresponding joint angles for desired end-effector positions
- Designed for generic tree or chain-structured manipulators



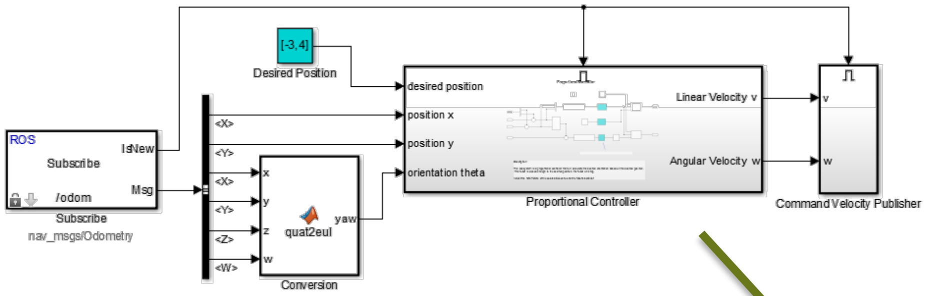
Baxter robot following pre-defined right hand trajectory

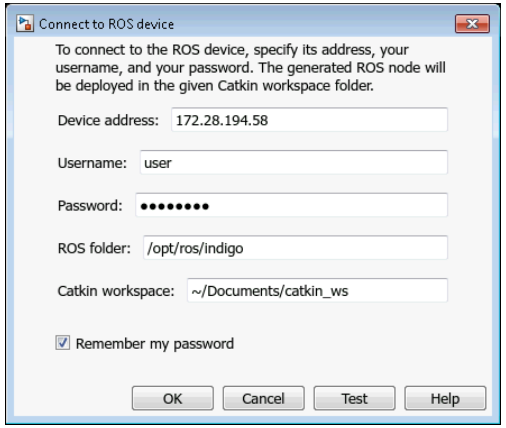
Using BFGS/gradient projection based IK algorithm

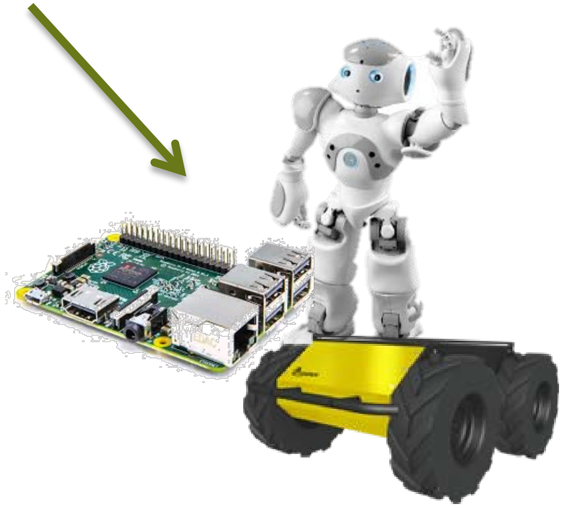
Automated Deployment of Simulink ROS Nodes

Automatically deploy ROS nodes to target hardware using Simulink Coder

- Automatically deploy and run ROS nodes using Simulink Coder
- Connect to ROS robot and deploy an executable ROS node for a Simulink model
- Validate device connection settings within Simulink
- Use the `rosdevice` object to connect to the target device and run or stop the deployed ROS nodes







```
>> device = rosdevice
```

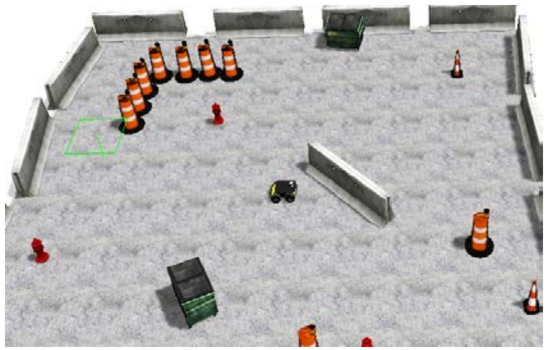
```
» robotROSCodeGenerationExample
```

Occupancy Grid Class

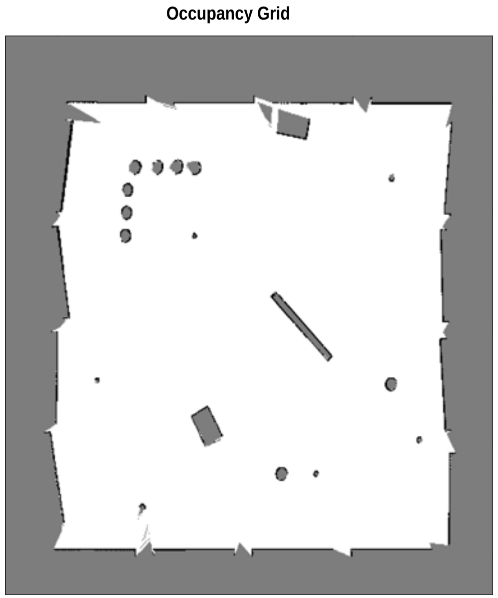
Build a robot environment using a 2D occupancy map with probabilistic values

- Create 2D occupancy maps using probabilistic values with `robotics.OccupancyGrid` class

- Use the occupancy grid with the `robotics.PRM` and `robotics.MonteCarloLocalization` classes for path planning and localization



Actual environment of the robot



Probabilistic representation using Occupancy Grid map

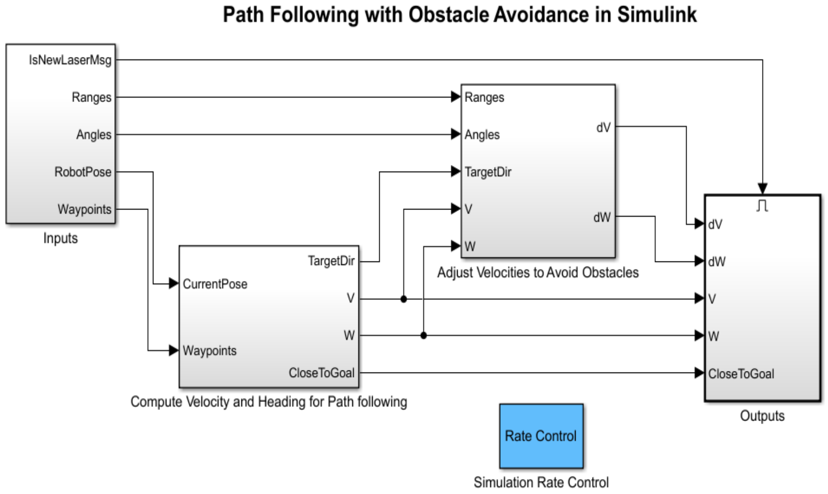
```
>> map = robotics.OccupancyGrid(20,20)

>> edit MappingWithKnownPosesExample
```

Mobile Robotics Algorithm Blocks

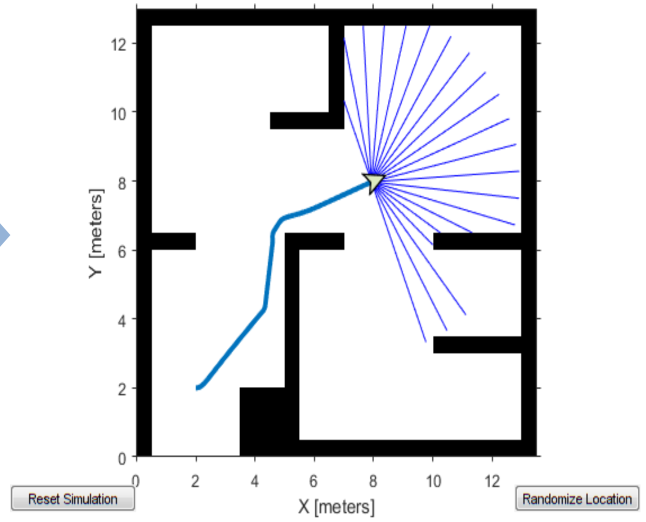
Perform obstacle avoidance and path following in Simulink

- Use the vector field histogram and Pure Pursuit algorithms with Simulink

- The `Pure Pursuit` block outputs a target direction, which you can feed directly into the `Vector Field Histogram` block to perform obstacle avoidance with path following



Simulink Model



Result: Path Following with Dynamic Obstacle Avoidance

```
>> robotalgslib
```

ROS Action Client

Send action goals via a ROS network and get feedback on their execution

- Perform predefined ROS network actions using `rosactionclient` function
- Send goal message to action server
- Cancel action goal at any time
- Wait until action server is available or until goal finishes execution
- Create custom callbacks for feedback and result messages

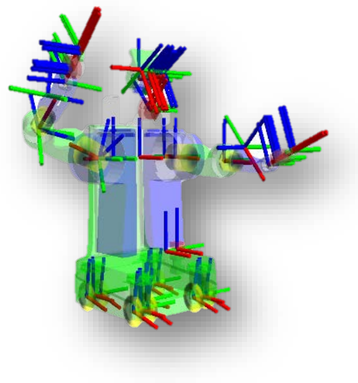
```
[tbot, goalmsg] = rosactionclient('/turtlebot_move',
'turtlebot_actions/TurtlebotMove')

% Wait for the action server to start up
waitForServer(tbot)


% Request forward movement and wait until the TurtleBot
% is done
goalmsg.ForwardDistance = 1.0;
resultmsg = sendGoalAndWait(tbot, goalmsg);
```

Buffered ROS tf Transformations

Access time-buffered transformations from the ROS transformation tree

- `getTransform` or `transform` to access and apply the transformations at a specified source time
- Interpolate transformations for requested time
- `canTransform` enables you to check if the transformation is available

» `ROSTransformationTreeExample`



```
% Create the transformation tree object.
>> tftree = rostf

tftree =

  TransformationTree with properties:

    AvailableFrames: {35x1 cell}
     LastUpdateTime: [1x1 Time]
         BufferTime: 10


% Get the transformation that was valid 1 second ago. Wait for
% up to 3 seconds for the transformation to become available.
>> tform = getTransform(tftree, 'base_link',
'camera_depth_frame', rostime('now') - 1, 'Timeout', 2)

tform =

  ROS TransformStamped message with properties:

     MessageType: 'geometry_msgs/TransformStamped'
          Header: [1x1 Header]
    ChildFrameId: 'camera_depth_frame'
       Transform: [1x1 Transform]

  Use showdetails to show the contents of the message
```

R2016b

ROS Time and Duration

Use mathematical operations on ROS time and duration objects

- Use `rostime` function to specify second and nanosecond scalar inputs when creating a `ROS Time` message object
- Use the new `rosduration` function to create a `ROS Duration` message object
- Support mathematical operations and comparisons

```
% Create time object from seconds and nanoseconds
>> t1 = rostime(1500,200000)

t1 =

  ROS Time with properties:
     Sec: 1500
    Nsec: 200000

% Create time object for total seconds
>> t2 = rostime(500.14671);

% Add 3 seconds to the time and calculate duration
% between two times
>> t2 = t2 + 3;
>> dur = t1 - t2

dur =

  ROS Duration with properties:
     Sec: 999
    Nsec: 853490000
```

Code Generation for Robotics Algorithms

Generate code for select algorithms with MATLAB Coder

- Code generation with MATLAB Coder is now available for the following algorithms:

```
robotics.BinaryOccupancyGrid
robotics.OccupancyGrid
robotics.OdometryMotionModel
robotics.PRM
robotics.PurePursuit
```