```
cmake_minimum_required(VERSION 2.8.3)
project(husky_highlevel_controller)

## Use C++11
add_definitions(--std=c++11)

## Find catkin macros and libraries
find_package(catkin REQUIRED COMPONENTS
  roscpp
  sensor_msgs
)

###################################
## catkin specific configuration ##
###################################
## The catkin_package macro generates cmake config files for your package
## Declare things to be passed to dependent projects
## INCLUDE_DIRS: uncomment this if you package contains header files
## LIBRARIES: libraries you create in this project that dependent projects also need
## CATKIN_DEPENDS: catkin_packages dependent projects also need
## DEPENDS: system dependencies of this project that dependent projects also need
catkin_package(
  INCLUDE_DIRS
    include
#  LIBRARIES
  CATKIN_DEPENDS
    roscpp
    sensor_msgs
#  DEPENDS
)

###########
## Build ##
###########

## Specify additional locations of header files
## Your package locations should be listed before other locations
include_directories(
  include
  ${catkin_INCLUDE_DIRS}
)

## Declare a C++ executable
add_executable(${PROJECT_NAME}
  src/${PROJECT_NAME}_node.cpp
  src/HuskyHighlevelController.cpp
)

## Specify libraries to link a library or executable target against
target_link_libraries(${PROJECT_NAME}
  ${catkin_LIBRARIES}
)
```

```
camera:
        left:
                name: left_camera
                exposure: 1
        right:
                name: right_camera
                exposure: 1.1
```

```
topic_name: /scan
queue_size: 10
```

```
<launch>
        <arg name="world" default="singlePillar"/>
        <include file="$(find husky_gazebo)/launch/husky_empty_world.launch">
        <arg name="world_name" value="$(find husky_highlevel_controller)/worlds/$(arg world).world"/>
        <arg name="laser_enabled" value="true"/>
        </include>

        <node name="husky_laserscan" pkg="husky_highlevel_controller" type="husky_highlevel_controller"
output="screen">

        <rosparam command="load" file="$(find husky_highlevel_controller)/config/localization.yaml"/></
node>
        <node name="rviz" pkg="rviz" type="rviz"/>


        <node pkg="pointcloud_to_laserscan" type="pointcloud_to_laserscan_node"
name="pointcloud_to_laserscan" output="screen"><remap from="cloud_in" to="camera/depth/points"/><remap
from="scan" to="camera/scan"/><rosparam>
            target_frame: base_link # Leave empty to output scan in the pointcloud frame
            tolerance: 1.0
            min_height: 0.05
            max_height: 1.0

            angle_min: -0.52 # -30.0*M_PI/180.0
            angle_max: 0.52 # 30.0*M_PI/180.0
            angle_increment: 0.005 # M_PI/360.0
            scan_time: 0.3333
            range_min: 0.45
            range_max: 4.0
            use_inf: true

            # Concurrency level, affects number of pointclouds queued for processing and number of
threads used
            # 0 : Detect number of cores
            # 1 : Single threaded
            # 2->inf : Parallelism level
            concurrency_level: 1
        </rosparam></node>






</launch>
```

```xml
<?xml version="1.0"?>
<!--
Software License Agreement (BSD)

\file      husky_empty_world.launch
\authors   Paul Bovbel <pbovbel@clearpathrobotics.com, Devon Ash <dash@clearpathrobotics.com>
\copyright Copyright (c) 2015, Clearpath Robotics, Inc., All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that
the following conditions are met:
 * Redistributions of source code must retain the above copyright notice, this list of conditions and the
   following disclaimer.
 * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the
   following disclaimer in the documentation and/or other materials provided with the distribution.
 * Neither the name of Clearpath Robotics nor the names of its contributors may be used to endorse or promote
   products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WAR-
RANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, IN-
DIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT
OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
-->
<launch>

   <arg name="world_name" default="worlds/empty.world"/>

  <arg name="laser_enabled" default="true"/>
  <arg name="ur5_enabled" default="true"/>
  <arg name="kinect_enabled" default="false"/>

  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name" value="$(arg world_name)"/> <!-- world_name is wrt GAZEBO_RESOURCE_PATH environment variable -->
    <arg name="paused" value="false"/>
    <arg name="use_sim_time" value="true"/>
    <arg name="gui" value="true"/>
    <arg name="headless" value="false"/>
    <arg name="debug" value="false"/>
  </include>

  <include file="$(find husky_gazebo)/launch/spawn_husky.launch">
    <arg name="laser_enabled" value="$(arg laser_enabled)"/>
    <arg name="ur5_enabled" value="$(arg ur5_enabled)"/>
    <arg name="kinect_enabled" value="$(arg kinect_enabled)"/>
  </include>

</launch>
```

```
// Work based off the open source course, Programming for Robotics - ROS
// by ETZH (http://www.rsl.ethz.ch/education-students/lectures/ros.html)
// Date:    3/29/2017
// Author:  Tasuku Miura

#include <ros/ros.h>
#include "husky_highlevel_controller/HuskyHighlevelController.hpp"

int main(int argc, char** argv)
{
  ros::init(argc, argv, "husky_highlevel_controller");
  ros::NodeHandle nodeHandle("~");

  husky_highlevel_controller::HuskyHighlevelController huskyHighlevelController(nodeHandle, false);

  ros::spin();
  return 0;
}
```

```
<launch>
        <arg name="world" default="singlePillar"/>
        <include file="$(find husky_gazebo)/launch/husky_empty_world.launch">
        <arg name="world_name" value="$(find husky_highlevel_controller)/worlds/$(arg world).world"/>
        <arg name="laser_enabled" value="true"/>
        </include>

        <node name="husky_laserscan" pkg="husky_highlevel_controller" type="husky_highlevel_controller" 
output="screen">
        <rosparam command="load" file="$(find husky_highlevel_controller)/config/default.yaml"/></node>
        <node name="rviz" pkg="rviz" type="rviz"/>

</launch>
```

```
// Work based off the open source course, Programming for Robotics - ROS
// by ETZH (http://www.rsl.ethz.ch/education-students/lectures/ros.html)
// Date:    3/29/2017
// Author:  Tasuku Miura

#include "husky_highlevel_controller/HuskyHighlevelController.hpp"
#include <cmath>

namespace husky_highlevel_controller {

    HuskyHighlevelController::HuskyHighlevelController(ros::NodeHandle& nodeHandle,
                                                       bool manual_control) :
        nh_(nodeHandle),
        husky_manual_control_(manual_control)
    {
        registerSubscriber();
        registerService();
        registerPublisher();
        ROS_INFO("Node launched.");
    }

    HuskyHighlevelController::~HuskyHighlevelController()
    {
    }

    void HuskyHighlevelController::registerService()
    {
        service_manual_control_ =
            nh_.advertiseService("manual_control_override", &HuskyHighlevelController::controlCB, this);
    }

    void HuskyHighlevelController::registerSubscriber()
    {
        sub_laser_scan_ =
            nh_.subscribe("/scan", 10, &HuskyHighlevelController::topicCB, this);
    }

    void HuskyHighlevelController::registerPublisher()
    {
        pub_husky_twist_ =
            nh_.advertise<geometry_msgs::Twist>("/husky_velocity_controller/cmd_vel", 10);
        pub_visualization_marker_ =
            nh_.advertise<visualization_msgs::Marker>("/husky_laserscan/visualization_marker", 0);
    }

    bool HuskyHighlevelController::controlCB(std_srvs::SetBool::Request &req,
                                             std_srvs::SetBool::Response &resp)
    {
        husky_manual_control_ = req.data;
        resp.success = true;
        return resp.success;
    }

    void HuskyHighlevelController::topicCB(const sensor_msgs::LaserScan& msg)
    {
        float min = INFINITY;   //Min value in range.
        float theta = 0.0;      //Turn angle.
        float d_init = 0.0;     //Initial distance to pillar.
        auto min_idx = 0;       //Index of distance of pillar.
        auto len = msg.ranges.size();

        geometry_msgs::TransformStamped transformStamped;
        try {
            transformStamped = tfBuffer_.lookupTransform("odom", "base_laser",
                                                         ros::Time(0));
        } catch (tf2::TransformException &ex) {
            ROS_WARN("%s", ex.what());
            ros::Duration(1.0).sleep();
        }

        for (auto i = 0; i < len; i++) {
            if (min == INFINITY)
                min = msg.ranges[i];

            if (msg.ranges[i] < min) {
                min = msg.ranges[i];
                min_idx = i;
            }
        }

        geometry_msgs::Twist cmd_msg;

        //Calculates turn angle, left hand rule.
        theta = msg.angle_min + min_idx * msg.angle_increment;
        cmd_msg.angular.z = -theta;

        //Stops Husky from crashing into pillar.
        if (min < 0.5)
            husky_manual_control_ = true;

        //Check if there has been trigger to stop Husky.
        if (!husky_manual_control_) {
            cmd_msg.linear.x = min;
            cmd_msg.linear.y = min;
        } else {
            cmd_msg.linear.x = 0.0;
            cmd_msg.linear.y = 0.0;
        }
        pub_husky_twist_.publish(cmd_msg);

        geometry_msgs::PoseStamped pose_in;
        geometry_msgs::PoseStamped pose_out;

        pose_in.pose.position.x = min*cos(-theta);
        pose_in.pose.position.y = min*sin(-theta);
        pose_in.header.stamp = ros::Time(0);
        pose_in.header.frame_id = "base_laser";
        tfBuffer_.transform(pose_in, pose_out, "odom");

        pillarMarker(pose_out.pose.position.x,
                     pose_out.pose.position.y);

        ROS_INFO("Distance to Pillar: %f", min);
    }

    void HuskyHighlevelController::pillarMarker(double x, double y)
    {
        visualization_msgs::Marker marker;
        marker.header.frame_id = "odom";
        marker.header.stamp = ros::Time();
        marker.ns = "husky_highlevel_controller";
        marker.id = 0;
        marker.type = visualization_msgs::Marker::SPHERE;
        marker.action = visualization_msgs::Marker::ADD;
        marker.pose.position.x = x;
        marker.pose.position.y = y;
        marker.pose.position.z = 1;
        marker.pose.orientation.x = 0.0;
        marker.pose.orientation.y = 0.0;
        marker.pose.orientation.z = 0.0;
        marker.pose.orientation.w = 1.0;
        marker.scale.x = 1;
        marker.scale.y = 1;
        marker.scale.z = 0.1;
        marker.color.a = 1.0;
        marker.color.r = 0.0;
        marker.color.g = 1.0;
        marker.color.b = 0.0;
        pub_visualization_marker_.publish(marker);
    }

} /* namespace */
```

```
// This file is the header to HuskyHighlevelController.cpp.
//
// Work based off the open source course, Programming for Robotics - ROS
// by ETZH (http://www.rsl.ethz.ch/education-students/lectures/ros.html)
// Date:    3/29/2017
// Author:  Tasuku Miura

#pragma once

#include <ros/ros.h>
#include <tf2_ros/transform_listener.h>
#include <sensor_msgs/LaserScan.h>
#include <geometry_msgs/Twist.h>
#include <geometry_msgs/TransformStamped.h>
#include <geometry_msgs/PoseStamped.h>
#include <tf2_geometry_msgs/tf2_geometry_msgs.h>
#include <std_srvs/SetBool.h>
#include <visualization_msgs/Marker.h>

namespace husky_highlevel_controller {

    /*
     *Class containing the Husky Highlevel Controller
     */
    class HuskyHighlevelController {
    public:
        /*
         *Constructor.
         */
        HuskyHighlevelController(ros::NodeHandle&, bool manual_control);

        /*
         * Destructor.
         */
        virtual ~HuskyHighlevelController();

    private:
        void registerService();
        void registerSubscriber();
        void registerPublisher();

        /*
         * Service that sets husky_manual_control, which allows
         * user to stop/start husky from command line using rosservice
         * call.
         * @args: req - request defined in std_srvs::SetBool.
         * @args: resp - response as defined in std_srvs::SetBool.
         * @rets: returns true on success.
         */
        bool controlCB(std_srvs::SetBool::Request &req,
                       std_srvs::SetBool::Response &resp);

        /*
         * Subscriber callback to calculate the distance to pillar,
         * and publishes geometry_msg::Twist to implement a
         * proportional controller.
         * @args: msg - contains info related to LaserScan.msg.
         */
        void topicCB(const sensor_msgs::LaserScan& msg);

        /* Defines location and related specification of marker
         * used to represent the pillar.
         * @args: x - x coordinate of pillar.
         * @args: y - y coordinate of pillar.
         */
        void pillarMarker(double x, double y);
    private:
        ros::NodeHandle nh_;
        ros::Subscriber sub_laser_scan_;
        ros::Publisher pub_husky_twist_;
        ros::Publisher pub_visualization_marker_;

        ros::ServiceServer service_manual_control_;
        bool husky_manual_control_;

        tf2_ros::Buffer tfBuffer_;
        tf2_ros::TransformListener listener_ {tfBuffer_};
        ros::ServiceClient client_;
    };

} /* namespace */
```

```
topic_name: /scan
queue_size: 10
topic_name: /velodyne/assembled_cloud_filtered
queue_size: 10
topic_name: /husky_velocity_controller/odem
queue_size: 10
topic_name: /imu/data
queue_size: 10
topic_name: /joint_states
queue_size: 10
topic_name: /odemetry/filtered
queue_size: 10





odom_frame: odom
base_link_frame: base_link
world_frame: odom

two_d_mode: true

frequency: 50

odom0: husky_velocity_controller/odom
odom0_config: [false, false, false,
               false, false, false,
               true, true, true,
               false, false, true,
               false, false, false]
odom0_differential: false
odom0_queue_size: 10

imu0: imu/data
imu0_config: [false, false, false,
              true, true, true,
              false, false, false,
              true, true, true,
              false, false, false]
imu0_differential: true
imu0_queue_size: 10
imu0_remove_gravitational_acceleration: true
```

```
<launch>
        <node name="name" pkg="package" type="node_type">
        <rosparam command="load"
                        file="$(find package)/config/config.yaml" />
        </node>
</launch>
```

```xml
<?xml version="1.0"?>
<package format="2">
  <name>husky_highlevel_controller</name>
  <version>0.1.0</version>
  <description>The husky_highlevel_controller package</description>
  <maintainer email="dominic.jud@mavt.ethz.ch">Dominic Jud</maintainer>
  <license>BSD</license>
  <author email="dominic.jud@mavt.ethz.ch">Dominic Jud</author>

  <buildtool_depend>catkin</buildtool_depend>

  <depend>roscpp</depend>
  <depend>sensor_msgs</depend>
</package>
```

```
<?xml version="1.0" ?>
<sdf version="1.4">
  <world name="default">
    <!-- A global light source -->
    <include>
      <uri>model://sun</uri>
    </include>
    <!-- A ground plane -->
    <include>
      <uri>model://ground_plane</uri>
    </include>
    <!-- Cylinder -->
    <model name='unit_cylinder'>
      <pose frame=''>20 5 0.5 0 -0 0</pose>
      <link name='link'>
        <inertial>
          <mass>1</mass>
          <inertia>
            <ixx>0.145833</ixx>
            <ixy>0</ixy>
            <ixz>0</ixz>
            <iyy>0.145833</iyy>
            <iyz>0</iyz>
            <izz>0.125</izz>
          </inertia>
        </inertial>
        <collision name='collision'>
          <geometry>
            <cylinder>
              <radius>0.2</radius>
              <length>2</length>
            </cylinder>
          </geometry>
          <max_contacts>10</max_contacts>
        </collision>
        <visual name='visual'>
          <geometry>
            <cylinder>
              <radius>0.2</radius>
              <length>2</length>
            </cylinder>
          </geometry>
          <material>
            <script>
              <name>Gazebo/Grey</name>
              <uri>file://media/materials/scripts/gazebo.material</uri>
            </script>
          </material>
        </visual>
        <self_collide>0</self_collide>
        <kinematic>0</kinematic>
      </link>
    </model>


  </world>
</sdf>
```

```
<?xml version="1.0"?>
<!--
Software License Agreement (BSD)

\file      gazebo_description.launch
\authors   Paul Bovbel <pbovbel@clearpathrobotics.com
\copyright Copyright (c) 2015, Clearpath Robotics, Inc., All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided
that
the following conditions are met:
 * Redistributions of source code must retain the above copyright notice, this list of conditions and the
   following disclaimer.
 * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and
the
   following disclaimer in the documentation and/or other materials provided with the distribution.
 * Neither the name of Clearpath Robotics nor the names of its contributors may be used to endorse or
promote
   products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED
WAR-
RANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR
PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
IN-
DIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT
OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
AND
ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
OTHERWISE)
ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
-->
<launch>

  <arg name="laser_enabled" default="true"/>
  <arg name="ur5_enabled" default="false"/>
  <arg name="kinect_enabled" default="false"/>
  <arg name="robot_initial_pose" default="$(optenv ROBOT_INITIAL_POSE)"/>
  <arg name="husky_gazebo_description" default="$(optenv HUSKY_GAZEBO_DESCRIPTION)"/>
  <arg name="ur5_control_yaml_file" default="$(find husky_control)/config/control_ur5.yaml"/>

  <param name="robot_description" command="$(find xacro)/xacro.py '$(arg husky_gazebo_description)'
    laser_enabled:=$(arg laser_enabled)
    ur5_enabled:=$(arg ur5_enabled)
    kinect_enabled:=$(arg kinect_enabled)
    " />

  <node name="robot_state_publisher" pkg="robot_state_publisher" type="robot_state_publisher" />

  <!-- Load Husky control information -->
  <include file="$(find husky_control)/launch/control.launch"/>

  <!-- Include ros_control configuration for ur5, only used in simulation -->
  <group if="$(arg ur5_enabled)">

    <!-- Load UR5 controllers -->
    <rosparam command="load" file="$(arg ur5_control_yaml_file)" />
    <node name="arm_controller_spawner" pkg="controller_manager" type="spawner" args="arm_controller --
shutdown-timeout 3"/>

    <!-- Fake Calibration -->
    <node pkg="rostopic" type="rostopic" name="fake_joint_calibration" args="pub calibrated std_msgs/Bool
true" />

    <!-- Stow the arm -->
    <node pkg="husky_control" type="stow_ur5" name="stow_ur5"/>

  </group>

  <group if="$(arg kinect_enabled)">

    <!-- Include poincloud_to_laserscan if simulated Kinect is attached -->
    <node pkg="pointcloud_to_laserscan" type="pointcloud_to_laserscan_node"
name="pointcloud_to_laserscan" output="screen">

        <remap from="cloud_in" to="camera/depth/points"/>
        <remap from="scan" to="camera/scan"/>
        <rosparam>
            target_frame: base_link # Leave empty to output scan in the pointcloud frame
            tolerance: 1.0
            min_height: 0.05
            max_height: 1.0

            angle_min: -0.52 # -30.0*M_PI/180.0
            angle_max: 0.52 # 30.0*M_PI/180.0
            angle_increment: 0.005 # M_PI/360.0
            scan_time: 0.3333
            range_min: 0.45
            range_max: 4.0
            use_inf: true

            # Concurrency level, affects number of pointclouds queued for processing and number of
threads used
            # 0 : Detect number of cores
            # 1 : Single threaded
            # 2->inf : Parallelism level
            concurrency_level: 1
        </rosparam>
    </node>

  </group>

  <!-- Spawn robot in gazebo -->
  <node name="spawn_husky_model" pkg="gazebo_ros" type="spawn_model"
    args="$(arg robot_initial_pose) -unpause -urdf -param robot_description -model mobile_base"/>


</launch>
```