Advanced Teleoperation Control System for PR2 Humanoid Robot

NSF Summer Undergraduate Fellowship in Sensor Technologies
Carlos Torres Casiano (Electrical Engineering) - University of Puerto Rico at Mayagüez
Jason C. Carter (Industrial Engineering) – Morgan State University
Advisor: Dr. Camillo J. Taylor

Abstract

The ultimate achievement in robotics is to build a robot that can perform perform a range of tasks by human. One such robot that is getting closer to this ability is the PR2 humanoid robot, from Willow Garage. The PR2 has capabilities that allow the user to program different movements in order to achieve multiple assignments. Being an open source robot, the PR2 allows users to read previous codes and manipulate them in order to complete new tasks. The purpose of this research is to output what the PR2 sees to the Head Mounted Display (HMD) worn by the user, and control the PR2's head with a motion tracking system (Vicon). The Vicon system captures the movement of markers placed along the HMD, and outputs an array of numbers that will go through a Python code in order to retrieve the necessary coordinates. These coordinates are sent to the PR2 which will enable it to move the head to the specific point in space. With all the components working together the user is capable of manipulating the PR2's sight for his own use.

Table of Contents

1. Introduction

In past decades robots have been developed for use in industry, war, and other areas to assist human needs. Robots are useful to society for completing tasks that are constantly repetitive or are dangerous for people. They can also help people who, because of disease or disability, are unable to perform some basic tasks. One such robot, the PR2, is a humanoid robot developed to complete simple tasks done by people.

The major aspect of this robot is the control system, ROS, used to implement and control the different sensors and cameras on the robot's body. These cameras and sensors let the user see what the robot is watching. Images captured by PR2 go through the ROS operating system and are sent to the computer. A recent addition to some control systems is a head-mounted display (HMD) worn by the operator. If an HMD is present, the input images of the computer output to the HMD. Continuous advancements in robotics have led to the need for newer and more powerful control systems. To improve the functionality of these new robots and give them more human-like abilities, more technology needs to be integrated into them. Typical robotic control uses a visual navigation system with a stationery camera. However, the user would then have to turn the whole robot to turn the camera. The goal of this research project is to build a device to allow a link between a user's head movement and the robot's head movement. Instead of using basic computer screens and panels to view what the robot sees, the user will wear an HMD and see what the robot sees in real time as well as control its motion.

In order to track the pan and tilt motion of the user wearing the HMD, a motion tracking system is required. This system would detect the pan and tilt motion of the HMD, and relay that information into a digital format so we can use it to control the robot's head. Previous studies by the University of Texas at Austin [1] have shown that a humanoid robot's body can be controlled with the use of a motion tracking system. Movement of a user's arms and legs will move the corresponding parts of the robot. Also, research completed at the University of Pennsylvania has shown the versatility and accuracy of using a motion tracking system [2] to control helicopters. For our research a series of cameras will pick up the motion of the HMD and relay its coordinates to a standby computer which will interpret the data for the ROS code.

Each of the following components will work together in a system to make this project possible. The HMD movements from the motion capture system will be mapped in the computer. When the user pans or tilts his head, a signal will be sent to the PR2 to move its head accordingly. The PR2 will interpret the data via an elaborate ROS code. Data images from the PR2 will be sent to the HMD worn by the user in real time.

2. Background

2.1 PR2

PR2 is a humanoid robot built and donated by Willow Garage to universities for research opportunities. Being a humanoid allows PR2 to accomplish small chores, and it can adapt to changes in its environment. The PR2 consists of a head, two arms with grippers, and a base. The head contains two stereo cameras with LED pattern projector, a 5MP camera, laser range finder and an inertial measurement unit (IMU). The forearms each contain an Ethernet-based, wide-angle camera, while the grippers have three-axis accelerometers and pressure sensor arrays on the fingertips and the base has a fixed laser range finder. The PR2 uses ROS, an open-source, meta-operating system in order to communicate with the computer.

PR2 is a ROS-based which allows it to be reprogrammable, able to complete different tasks like folding towels, plugging cables, playing pool and opening doors among other things. Using the ROS system, PR2 is able to employ complex algorithms, being capable of attacking a problem continuously until it can achieve the goal. For example trying to plug its power cable to an outlet, PR2 continues to calculate to reposition its arm until achieving the goal. It is also capable of using sensors to generate a 3D map of its surroundings, as well as promptly compute unexpected changes in the environment.

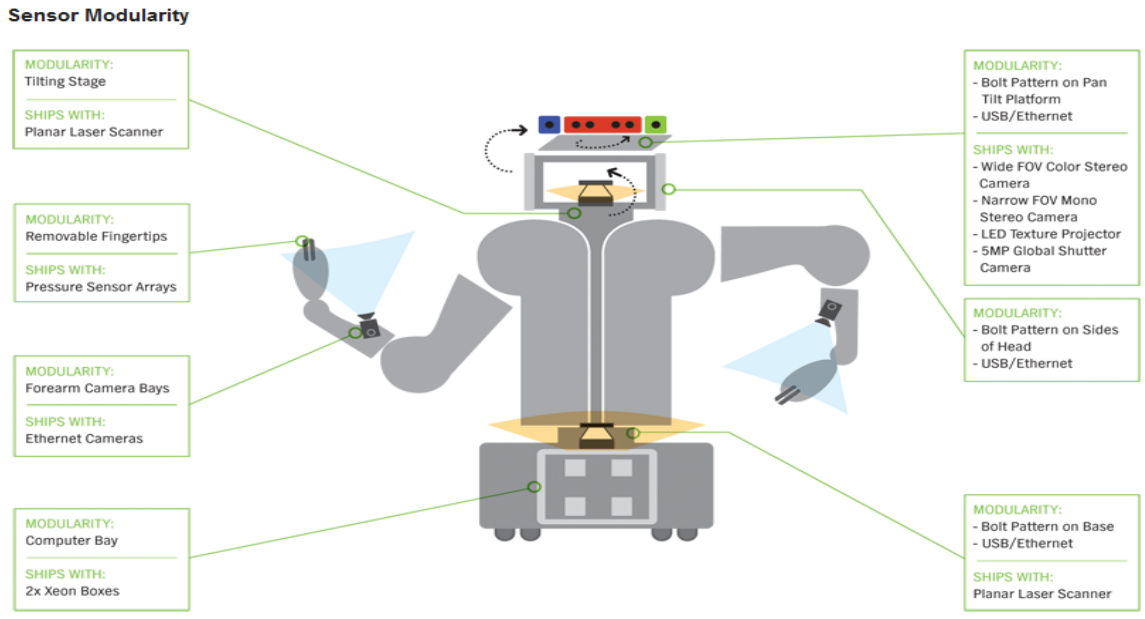


Figure 1: PR2 Sensor Modularity [3]

2.2 Motion Capture System

A motion capture camera, in this case Vicon, allows the user to record movement of people or objects and digitalized into a computer. This animation data is mapped to a 3D model to obtain the exact movement of the actor. Special markers that reflect infrared light are attached to the object by the user and then picked up using multiple cameras in the system, infrared light is then sent to the points attached and reflected to the cameras where they can map the form and position. This digital simulation can be used to draw a path for the programmable object, to study human movements, and to create movies, video games, etc.

Vicon Motion Tracking System

Infrared Camera

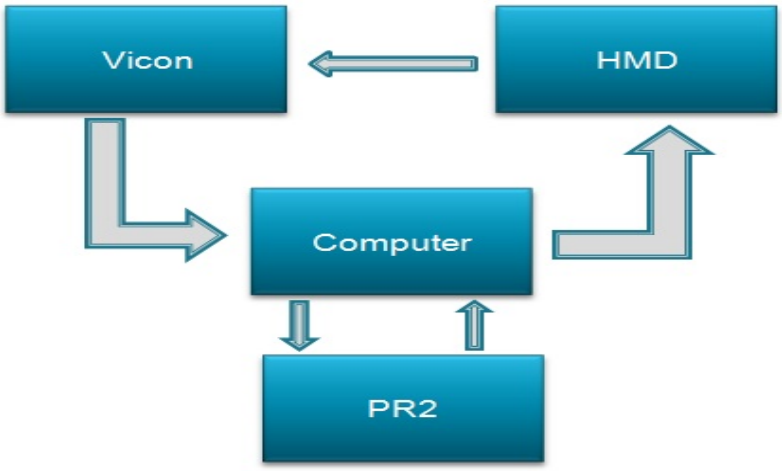
2.3 Head Mounted Display (HMD)

Using the Vuzix Wrap 920, an HMD with two displays in each lens, the user can see in real time the exact thing PR2 is watching. This is a key object in the research since the points of the Motion Caption System will be placed along the HMD.

Vuzix Wrap 920

3. Materials & Methods

All three major components in this project, PR2, HMD and Vicon Motion Capture System, are well synchronized to have the desired outcome. Each of these components communicate with one another through a series of computer code. The Vicon system receives the points of the HMD through its system, the points are then sent to the PR2 wirelessly through a computer, the PR2 sends its video output to the HMD.



3.1 PR2

The PR2, a humanoid robot made by Willow Garage, has various functions and abilities. We are mainly focused on its head movement and the camera inside. Using an open source code for head movement (Fig. 2), we were able to manipulate the head so that we could input our own movements. In lines 19, 20, and 21, there are movements for the associated X, Y, and Z motions. The inputs for the code came from the Vicon System and were compiled on the code we made (FIG. 1), and the coordinates generated from that were imported into the ROS open source code for the PR2.

3.2 Vicon Motion Tracking System

The motion tracking system uses a series of infrared cameras that captures the position of sets of markers within its range. The HMD used for this project was aligned with markers along the frame, and the Vicon system picked up the markers and displayed a frame of the HMD on the computer's screen. Along with these points on the screen from the HMD, the Vicon system displayed a series of points where the glasses were in space. The system continuously displayed these points as the glasses moved through the Vicon system

3.3 Vuzix Wrap 920

In order to receive images from what the robot sees, we decided to use the Vuzix Wrap 920 as the head mounted display. The Vuzix Wrap supplies vision to both eyes of the user in a resolution of 680 x 480, and is equipped with USB and VGA connections. The PR2 is connected to a computer, and the PR2's head camera is shown on the computer. With the supplied connectors, the Vuzix is capable of displaying what is on the computer screen, like an external monitor. The user is then able to see what the robot sees through this manner.

3.4 Programming Code

A. Vicon

The first step for the programming code is to pull the values from the Vicon System. The system outputs a series of "tupled" data, and we only need to pull out a series of values from that data. This code connects the external computer to the Vicon System and pulls out the data that we need (Fig. 5). The data is a set of values that relate to the rotation of the HMD within the system. The values are displayed in radians from the center axis of the Vicon and the HMD (Fig 8).

B. Vicon to PR2

The pulled values then need to be converted into something the PR2 can understand. This code takes the pulled numbers, converts them to degree movements to the corresponding X, Y, or Z movement, and puts it into a ROS code for the PR2 (Fig. 6). As the HMD is rotated through its axis, pan motion, X and Y, and tilt motion, Y and Z, are relayed to the PR2.

C. PR2 to HMD

The PR2 has a camera inside of its head unit. Through one of the ROS codes we can communicate with the camera and display what the PR2 views onto the computer screen. This code allows for that communication and displays the images from the PR2 onto the fullscreen of the computer (Fig. 7). Since the HMD works as an external monitor connected to the computer, the fullscreen view gives the user a complete view from the PR2 and not just in a specified window.

4. DISCUSSION & RESULTS

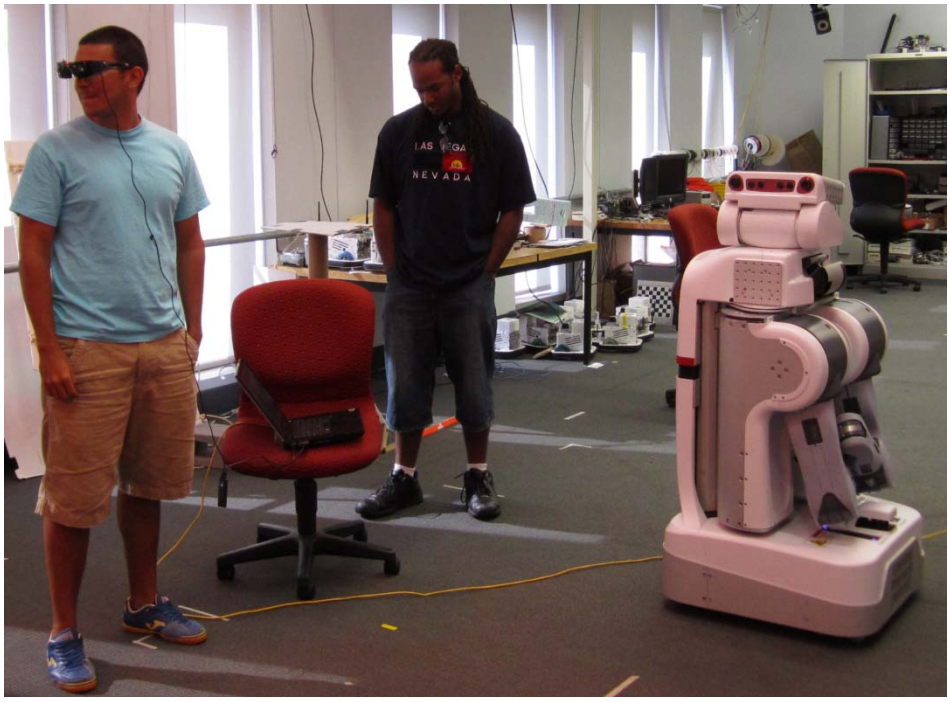
The purpose of this research project was to create a way to control the head motion of a robot, the PR2, only by moving one's head, as well as to see what the robot is seeing. The idea was to use a Head Mounted Display to see what the robot sees, and then use a motion tracking system to send the relative points of the user moving his head to the robot. This is an ongoing study that in the end, the idea is to be able to have a robotic avatar; to program the robot to essentially mimic the movements of the user. This setup was just the beginning with vision and controlling from the head. In order for full body movement, the next task would be to generate codes for arms and velocity directions for the robot, and still using the motion tracking system to receive points from the user's body to input into the code.

Upon conclusion of this project, we were able to control the PR2's head with basic pan and tilt motions of our head. While wearing the glasses we were able to locate objects around the PR2 while in a different location. The idea was to test the capabilities of the HMD and the PR2. The user would search the room that the PR2 was in and locate an object that was previously shown to him, and then describe where it was located. Another task was to read a small selection in front the PR2 that the viewer could see through the HMD he was wearing. These two tasks tested the capability of the completed project, how well it works, and how clearly the vision is at particular distances.

Results- Controlling the Head

Along the journey of completing this project there have been a few challenges. One of the biggest was connecting the entire system together. The PR2 and Vicon System needed to be connected on the same system in order to communicate the points in the code. A set of concurrent IP addresses needed to be made for the two components, and a computer connected to the Vicon system was used to integrate them together. A separate computer was also needed to connect the PR2 to the HMD and the problem was running Ethernet cables to the PR2 to both and having a dedicated network setup for everything to run on. Another challenge was the creation of the code that would take data from the Vicon system, compile it, and sent it to the PR2 for head moment. We first had to learn how the Python and Emacs programs work. Upon learning the scripts, we were able to generate a code that takes two different angle degrees, first for XY coordinate, second for YZ coordinate, as well as a height set for the height of the PR2, and then compiles the sets to determine where the PR2 would look.

The code that communicates the motions of the HMD worn by the user through the Vicon System to the PR2 does so through a series of steps. The Vicon generates a value of movement, the PR2 receives the value modified from the code, the PR2 then executes the motion, and this cycle is repeated continuously. The problem is that when a movement by the user does not coincide with the a movement that the PR2 can make, such as a complete ninety degree turn of the head up or down, the cycle is not complete and the PR2 “locks” and exits the code.



Controlling Head

5. Acknowledgments

We would like to recognize several people and organizations for making this all possible this summer. SUNFEST and Dr. Jan Van Der Spiegel made it possible for us to be here and participate in the research. As well as the Electrical Systems Engineering here at U Penn, for hosting the SUNFEST program. In addition to U Penn, we would like to thank the National Science Foundation for the grant funding the program as well as ARTSI too.

We would also like to thank Dr. C.J. Taylor who guided us through this whole endeavor, as well as his graduate student Anthony Cowley, who helped us learn the software that we needed in order to complete the tasks.

6. References

[1] Adam Setapen, Michael Quinlan, and Peter Stone, *MARIOnET: Motion Acquisition for Robots through Iterative Online Evaluative Training*. May 2010. University of Texas at Austin.

[2] D. Mellinger, "Quadrotor", fling.seas.upenn.edu. June 11, 2010. [Online]. Available: http://fling.seas.upenn.edu/~dmel/wiki/index.php?n=Main.Quadrotor. [Accessed June 29, 2010]

[3] Willow Garage, "PR2 Overview", willowgarage.com. Copyright © 2008-2010. [Online]. Available: http://www.willowgarage.com/pages/robots/pr2-overview. [Accessed June 29, 2010]

[4] ROS Wiki, "Documentation", ros.org. Licensed under 'Creative Commons Attribution 3.0' [Online] Available: http://www.ros.org/wiki/. [Last Edited June 3, 2010]

7. Appendix

Fig. 1 Code to find x y z input

Fig. 2 ROS code for PR2

Fig. 3

Fig. 4 Vicon Code

Fig. 5 Vicon to PR2 code

Fig. 6 PR2 to HMD code

Fig. 7 HMD axis view from Vicon

Code to find x, y, z coordinates from angle inputs:

```
from math import *
def locate(az, el, height):
    pan = radians(az)
    tilt = radians(el)
    Y = cos(tilt) * cos(pan)
    X = -cos(tilt) * sin(pan)
    Z = sin(tilt) + height


    print ' X: '+ str(X) + '\n Y: ' + str(Y) + '\n Z: ' + str(Z)
    return (X,Y,Z)
```

FIG 1. 3D CODE; DEGREES à COORDINATE

ROS code to control the head of PR2:

```
1 #! /usr/bin/python
 2 import roslib
 3 roslib.load_manifest('mm_teleop')
4
5 import rospy
6 import actionlib
7 from actionlib_msgs.msg import *
8 from pr2_controllers_msgs.msg import *
9 from geometry_msgs.msg import *
10
```

```
11 rospy.init_node('move_the_head', anonymous=True)
12
13 client = actionlib.SimpleActionClient(
14     '/head_traj_controller/point_head_action', PointHeadAction)
15 client.wait_for_server()
16
17 g = PointHeadGoal()
18 g.target.header.frame_id = 'base_link'
19 g.target.point.x = 1.0
20 g.target.point.y = 0.0
21 g.target.point.z = 1.0
22 g.min_duration = rospy.Duration(1.0)
23   24 client.send_goal(g)
25 client.wait_for_result()
26
27 if client.get_state() == GoalStatus.SUCCEEDED:
28     print "Succeeded"
29 else:
30     print "Failed"
```

FIG 2 ROS CODE (We still need to edit to add links for lines 19, 20, 21)

Vicon Code

```
#include <string>
#include <vector>
#include <ros/ros.h>

#include <vicon/Names.h>
#include <vicon/Values.h>
#include "ViconDriver.h"
```

```
ros::Publisher pub_names;
vicon::Names names_msg;

void names_callback(const ros::TimerEvent& e)
{
  pub_names.publish(names_msg);
}

int main(int argc, char** argv)
{
  ros::init(argc, argv, "vicon");

  ros::NodeHandle n("~");

  pub_names = n.advertise<vicon::Names>("names", 100, true);

  ros::Publisher pub_values
    = n.advertise<vicon::Values>("values", 100);

  vicon::Values values_msg;
  n.param("frame_id", values_msg.header.frame_id,
          string("vicon"));
  const int buff_len = 1024;
  const int num_buff = 10;
  const bool stream = true;
  ViconDriver vd(buff_len, num_buff, stream);

  vector<double> values;
  vector<string> names;

  string server;
  n.param("server", server, string("alkaline"));
  int port;
  n.param("port", port, 800);

  // Connect
  if (vd.ConnectTCP(server.c_str(), port))
    {
      ROS_FATAL("%s: failed to connect to %s:%i",
                ros::this_node::getName().c_str(),
                server.c_str(), port);
      return -1;
    }

  // Start device
  if (vd.StartDevice() != 0)
    {
      ROS_FATAL("%s: could not start device",
                ros::this_node::getName().c_str());
```

```
    return -1;
  }

//get names (buffered in the driver)
if (vd.GetNames(names) != 0)
  {
    ROS_ERROR("%s: could not get names",
              ros::this_node::getName().c_str());
    return -1;
  }

names_msg.names.resize(names.size());
std::copy(names.begin(), names.end(),
          names_msg.names.begin());

values_msg.values.resize(names.size());

ros::Timer timer = n.createTimer(ros::Duration(1),
                                 names_callback);

while (n.ok())
  {
    //get values
    if (vd.GetValues(values) != 0)
      {
        ROS_INFO("%s: could not get values",
                 ros::this_node::getName().c_str());
        continue;
      }

    if (names.size() != values.size())
      {
        ROS_INFO("%s: names size does not match values size",
                 ros::this_node::getName().c_str());
        continue;
      }

    std::copy(values.begin(), values.end(),
              values_msg.values.begin());

    values_msg.header.stamp = ros::Time::now();

    pub_values.publish(values_msg);

    ros::spinOnce();
  }

if (vd.StopDevice() != 0)
  {
    ROS_FATAL("%s: could not stop device",
```

```
                ros::this_node::getName().c_str());
      return -1;
    }

  if (vd.Disconnect())
    {
      ROS_FATAL("%s: failed to disconnect",
                ros::this_node::getName().c_str());
    }

  return 0;
}
```

Fig. 5 Vicon Code

Vicon to PR2 Code

```
#!/usr/bin/env python
import roslib;
roslib.load_manifest('vicon_hmd')

from vicon.msg import Values
import rospy
from math import *

import actionlib
from actionlib_msgs.msg import *
from pr2_controllers_msgs.msg import *
from geometry_msgs.msg import *

# Global Constants

# Robot height - should be gleaned from robot
HEIGHT = 1.36

# Filtering constant
ALPHA = 0.5

# Maximum motion in degrees per command period
MAX_DELTA = 10

def rad2deg(rad):
    return ((rad/pi)*180.0)

def deg2rad(deg):
    return ((deg/180.0)*pi)

def clip (x, clip):
    if (x > clip):
        return clip
    elif (x < -clip):
```

```
        return -clip
    else:
        return x

# Filtered version of angles returned by Vicon - global variables
ax_f, ay_f, az_f = 0.0, 0.0, 0.0

def callback(data):
    global ax_f, ay_f, az_f

    # Get current values of angles and update filter
    ax_f = ALPHA*ax_f + (1-ALPHA)*(data.values[-6])
    ay_f = ALPHA*ay_f + (1-ALPHA)*(data.values[-5])
    az_f = ALPHA*az_f + (1-ALPHA)*(data.values[-4])

def listener():
    rospy.init_node('vicon_hmd_listener')
    rospy.Subscriber("/vicon/values", Values, callback)

    # Robots azimuth and elevation angles
    az, el = 0.0, 0.0

    client = actionlib.SimpleActionClient(
        '/head_traj_controller/point_head_action', PointHeadAction)
    client.wait_for_server()

    g = PointHeadGoal()
    # g.target.header.frame_id = 'head_plate_frame'
    g.target.header.frame_id = 'base_link'
    g.min_duration = rospy.Duration(0.1)

    while not rospy.is_shutdown():
        # Main loop that sends commands to the robot

        # Come up with target azimuth and elevation
        delta_az = clip(az_f - az, deg2rad(MAX_DELTA))
        az = clip (az+delta_az, deg2rad(90))

        delta_el = clip(ax_f - el, deg2rad(MAX_DELTA))
        el = clip(el+delta_el, deg2rad(45))

        # Send the command to the PR2

        g.target.point.x = cos(el)*cos(az)
        g.target.point.y = cos(el)*sin(az)
        g.target.point.z = sin(el) + HEIGHT

        client.send_goal(g)
        client.wait_for_result()
```

```
        # Changing the sleep time changes the rate at which commands
are sent
        rospy.sleep (0.1)

if __name__ == '__main__':
    listener()
```

Fig. 6 Vicon to PR2

PR2 to HMD Code

```
#!/usr/bin/env python
import roslib; roslib.load_manifest('FullScreen')
import rospy
from sensor_msgs.msg import CompressedImage
import pygame

# The PyGame screen object
screen = None

# Screen reoslution
screen_size = (0,0)

# Flag to indicate when the user has requested we shutdown
running = True

def save_image(msg):
    """Save an image to disk. Return the file name."""
    if msg.format == "jpeg":
        tmp_name = "/tmp/pr2cam.jpeg"
    else:
        tmp_name = "/tmp/pr2cam.png"

    tmp_file = open(tmp_name,'wb')
    tmp_file.write(msg.data)
    tmp_file.close()
    return tmp_name

def show_image(msg):
    """Show an image fullscreen using PyGame."""
    global screen, screen_size, running

    # Save the image to a temporary file
    tmp_name = save_image(msg)

    # Load the temporary image into PyGame
    try:
        frame = pygame.image.load(tmp_name).convert()
        pygame.transform.scale(frame, screen_size, screen)
        pygame.display.update()
```

```
    except Exception as e:
        if running:
            raise e

def init_pygame():
    """Initialize PyGame and get screen resolution."""
    global screen, screen_size
    pygame.init()
    screen = pygame.display.set_mode(screen_size,
                                     pygame.HWSURFACE |
                                     pygame.DOUBLEBUF |
                                     pygame.FULLSCREEN)
    screen_size = screen.get_size()

def handle_events():
    """Return False when the user hits the escape key."""
    for event in pygame.event.get():
        if event.type == pygame.KEYDOWN and event.key ==
pygame.K_ESCAPE:
            return False
        elif event.type == pygame.QUIT:
            return False
    return True

if __name__ == '__main__':
    try:
        init_pygame()
        rospy.init_node('FullScreen')
        default_topic =
'/wide_stereo/right/image_rect_color/compressed'
        cam_topic = rospy.get_param('~cam', default_topic)
        rospy.Subscriber(cam_topic, CompressedImage, show_image)

        while handle_events() and not rospy.is_shutdown():
            rospy.sleep(0.5)

        running = False
        pygame.quit()

    except rospy.ROSInterruptException:
        pass
```

Fig. 7 Fullscreen HMD Code from PR2