Eye

## EyeData.srv

## RobotControl.srv

Robot  
Controller

VirtualEngCont.srv

Virtual  
Engine

HUB

## 

## MasterControl.srv PlayerControl.srv

Player

GameMaster

**Figure.1:** Overall System Architecture

## Arrow heads imply that node is the client node of that service.

**TUTORIALS**

**What does .msg and .srv files include ?**

**.SRV FILES**

**MasterControl.srv**

string ID

string CMD

string gameMode

Map map

Player[] players

----

string Status

# Reserved Map

# 0 of the map is altitude map

# 1 of the map is friction map

# 2 of the map is temperature map

**EyeData.srv**  
string ID

uint8 cmd

uint16[] size

---

Map background

Vector4[] robots

Vector4[] objects

# cmd=1 -> take background image

# cmd=2 -> find robot and object

# robots are defined as x,y,theta coordinates and sticker ID

# objects are defined as x,y and w,h of the rectangular borders

**RobotControl.srv**  
Player[] players

---

uint8 error

# RobotSpeed=u,v,omega,ID

**VirtualEngCont.srv**  
string ID

string CMD

Map sceneMap

Player[] players

---

Player[] players

**PlayerControl.srv**

string CMD

Player myPlayer

----

uint8 Status

Player yourPlayer

Map map

**.MSG FILES**

**Sensor.msg**  
string name

string ID

Vector4 relativeHeading

float32 range

float32 angularRange

uint16 targetSticker

float32[] data

**Player.msg**

string ID

uint8 Sticker

Sensor[] sensors

string playable

float32 speedCoefficient

Vector4 position

Vector4 controlCommand

Vector4 modifiedCommand

**Flatten2DArray.msg**

float32[] Array

**Map.msg**

# Size is 1x2 tuple

uint16[] size

flatten2DArray[] parallelMap

**Vector4.msg**

float32 x

float32 y

float32 z

float32 w

**How to Pair Bluetooth Devices With Ubuntu?**

**Check Available**

hcitools scan

**Add Device**

bluez-single-agent hci0 XX:XX:XX:XX:XX:XX

XX:XX:XX:XX:XX:XX = bdaddr

**Trust**

bluez-test-device trusted XX:XX:XX:XX:XX yes

**Connect**

bluez-test-input connect XX:XX:XX:XX:XX yes

**Remove (if necessary)**

bluez-test-device remove XX:XX:XX:XX:XX

**To List Paired Devices (if necessary)**

bluez-test-device list

**How to Calibrate Robots ?**

Using RobotControl.py, send commands to the robot check whether it satisfies the conditions below:

**0 0 0** : Robot should stay still, no rotation no linear motion. If not, modify midVals accordingly.

**1 0 0** : Robot should move on a straight line in +x direction. If not modify

maxVals accordingly.

**-1 0 0** : Robot should move on a straight line in –x direction. If not modify

minVals accordingly.

**0 1 0** : Robot should move on a straight line in +y direciton. If not modify

maxVals accordingly.

**0 -1 0** : Robot should move on a straight line in -y direciton. If not modify

minVals accordingly.

**0 0 1** : Robot should turn about itself.

**Robot Control Setup**

***WARNING: Make sure the bluetooth addresses of robots and addresses in the code match !***

We have two built-in code for controling purposes. First one is RobotControl.py is used as following

py RobotControl [optional: Sticker] [optional :V\_x V\_y V\_z]

Sticker : 3,4

V\_x,y,z : -1 – 1 (in this range)

usage : you can use for calibration or games with one robot

Second one is RobotControl\_Two\_player.py

py RobotControl\_Two\_player.py

usage : you can use for games with players

**How to Write A Game Master?**

***WARNING : MasterName should be the same with the one on Hub.py&Eye.cpp***

**ROS**

rospy.wait\_for\_service('MasterControlService')

Hub\_Service= rospy.ServiceProxy('MasterControlService',MasterControl)

**Player Initiation**

request=MasterControlRequest()

request.ID="MasterName"

request.CMD="Init"

request.players=[Player()]

request.players[0].Sticker=3

request.players[0].playable="yes"

request.players[0].speedCoefficient=0.5

**Target Initiation**

request.players.append(Player())

request.players[1].Sticker=7

#Targets are non-playable

request.players[1].playable="no"

**Sensor Initiation**

# Each sensor has its own parameters that needs to be filled. Other parameters which are not used in Sensor.msg does not need to be filled.

#Target Sensor: Follows a static and dynamic target.

request.players[0].sensors.append(Sensor())

request.players[0].sensors[0].name="Target\_Sensor"

request.players[0].sensors[0].ID="Sensor1"

request.players[0].sensors[0].targetSticker=7

# GPS sensor

request.players[0].sensors.append(Sensor())

request.players[0].sensors[1].name="GPS"

# Sharp sensor

request.players[0].sensors.append(Sensor())

request.players[0].sensors[0].name="Sharp "

request.players[0].sensors[0].ID="Sensor2"

request.players[0].sensors[0].relativeHeading.z=0

request.players[0].sensors[0].range= 90

#LiDAR sensor

request.players[0].sensors.append(Sensor())

request.players[0].sensors[0].name="LiDAR "

request.players[0].sensors[0].ID="Sensor3"

request.players[0].sensors[0].range= 100

request.players[0].sensors[0].angularRange= 100

#Ultrasonic sensor

request.players[0].sensors.append(Sensor())

request.players[0].sensors[0].name="Ultrasonic "

request.players[0].sensors[0].ID="Sensor4"

request.players[0].sensors[0].range= 50

request.players[0].sensors[0].angularRange= 50

**Setting Game**

Hubresponse=Hub\_Service(request)

waiting=True

while waiting:

request= MasterControlRequest()

request.ID="MasterName"

request.CMD="IsReady"

Hubresponse=Hub\_Service(request)

if Hubresponse.Status=="Ready":

waiting=False

else:

time.sleep(0.5)

#Send start message

request= MasterControlRequest()

request.ID="MasterName"

request.CMD="Start"

Hubresponse=Hub\_Service(request)

**How to Write A Player?**

**ROS**

**Be careful about the speed of your hub request, a code pending player control services may result lag in Hub**

rospy.wait\_for\_service('PlayerControlService')

playerControlService = rospy.ServiceProxy('PlayerControlService',\ PlayerControl)

self.TERMIOS=termios # For keyboard input

**Player Initiation**#Here code initializes player nodes, let them chose their robot, drive robot

with keyboard inputs and collect their sensor data from the Hub node.

#Fill in the player request and initialize your player

self.CMD = 'Init'

self.myPlayer = Player()

self.myPlayer.ID = 'PlayerName'

self.myPlayer.Sticker = sticker

self.myVelocity = Vector4()

#Request your Robot from the Hub

HubRequest = PlayerControlRequest(self.CMD,self.myPlayer)

HubResponse = playerControlService(HubRequest)

while HubResponse.Status ==10:

#Keep checking whether the game has started or not.

self.CMD = 'Start?'

HubRequest = PlayerControlRequest(self.CMD,self.myPlayer)

HubResponse =playerControlService(HubRequest)

time.sleep(0.5)

while HubResponse.Status == 0:

#No problem, game can start.

self.CMD = 'Sensor'

HubRequest = PlayerControlRequest(self.CMD,self.myPlayer)

HubResponse = playerControlService(HubRequest)

#Here you can play! You can use keyboard input too!

input=self.getkey()s

if input=="w":

#forward

self.myVelocity.x=1

self.myVelocity.y=0

self.myVelocity.z=0

elif input=="s":

#backward

self.myVelocity.x=-1

self.myVelocity.y=0

self.myVelocity.z=0

elif input=="a":

#turn about itself

self.myVelocity.x=0

self.myVelocity.y=0

self.myVelocity.z=-1

elif input=="d":

#turn about itself the other way

self.myVelocity.x=0

self.myVelocity.y=0

self.myVelocity.z=1

else:

#stop

self.myVelocity.x=0

self.myVelocity.y=0

self.myVelocity.z=0

input=""

time.sleep(0.05)

self.CMD = 'Control'

self.myPlayer.controlCommand = self.myVelocity

#Send your request to the Hub

HubRequest = PlayerControlRequest(self.CMD,self.myPlayer)

HubResponse = playerControlService(HubRequest)

def getkey(self):

#getting keyboard input

fd = sys.stdin.fileno()

old = termios.tcgetattr(fd)

new = termios.tcgetattr(fd)

new[3] = new[3] & ~self.TERMIOS.ICANON & ~self.TERMIOS.ECHO

new[6][self.TERMIOS.VMIN] = 1

new[6][self.TERMIOS.VTIME] = 0

termios.tcsetattr(fd, self.TERMIOS.TCSANOW, new)

c = None

try:

c = os.read(fd, 1)

finally:

termios.tcsetattr(fd,self.TERMIOS.TCSAFLUSH, old)

return c

if \_\_name\_\_=='\_\_main\_\_':

Sticker=int(sys.argv[1])

if (type(Sticker)==int):

user1=playerClient(Sticker)

**Example Codes**GM\_Example\_Wanderer : Keyboard controlled single agent.

GM\_Example\_Targeter : With usage of GPS sensor or TargetSensor, one can follow both static and dynamic targets.

GM\_Example\_CollisionAvoider : With two onboard Sharp sensors, one at the front and one at the left, robot avoids collision with walls and obstacles.

GM\_Example\_Predator\_Prey : Two agent game, where one of the agents is set as a target of the other one and prey should be controlled by user from RobotArenaKeyboardControl.  
  
Complementing to these we have also player codes available as PL\_example\_CollisionAvoider, PL\_example\_Follower and RobotArenaKeyboardControl.

**How to Start A Game?**

**System Requirements : openCV 3.0.0.rc1, ROS indigo or newer versions, bluez bluetooth manager, Ubuntu 14.04 Trusty or newer version. Trusty may have problems with ROS indigo in terms of xserver failures.**

1. **roscore**
2. **rosrun without Eye  
     
   *#while running python codes check that you are at the correct directory  
   #Be careful that robots are not on the scene at this stage. Wait for it..***
3. **python VirtualEngine.py**
4. **python RobotController.py**

**OR  
python RobotController\_Two\_Player.py**

1. **python GM\_XXX.py #name your own GM  
     
   *#If no error is received, at this stage Hub is waiting for players to #connect!  
   #Please insert your robots to the scene now!***
2. **python PL\_XX.py #name your own Player**

**How to Create Custom Sensors?**

- For a new sensor which uses Map.parallel[0], which is the position map of the walls and obstacles, an additional and proper *if structure* to the VirtualEngine mySensor will be enough.

- If you want to create your own sensors (which we do not recommend at all),

GM needs to arange a new map with the same size of the position map, containing the new data you want to collect with your sensors. This map should be initiated in Hub, under “Init” as self.Map.parallel[1] = myNewMap. Then the new sensors in VirtualEngine should work with this new map. Moreover, if robots will react to the sensor data, myVelocity calculations are also needed to be updated according to the new map.

- In the Hub.py under “Init” do a binary summation the map which GM sends with the self.Map.parallel[0] and continue.

- paralelMap[0] of Map() data is reserved for virtual walls and obstacles. If you change this map, it will only create trouble.

**TORUBLESHOOT:**

**#If Roscore is not working Try:**

**export ROS\_HOSTNAME=localhost**

**export ROS\_MASTER\_URI=http://localhost:11311**

**#If .srv or .msg files are modified, you need to check first CMakeLists.txt #and then**

**source ~/catkin\_ws/devel/setup.bash**

**#Black marks or stains around the board disturbs the Eye, ROI Error!**

**#If this happens, use thresholded image or adjust the board.  
(Don’t adjust too hard though, camera is not fixed to the frame)**