Introduction to Game Programming and Robotics

Unit # 10

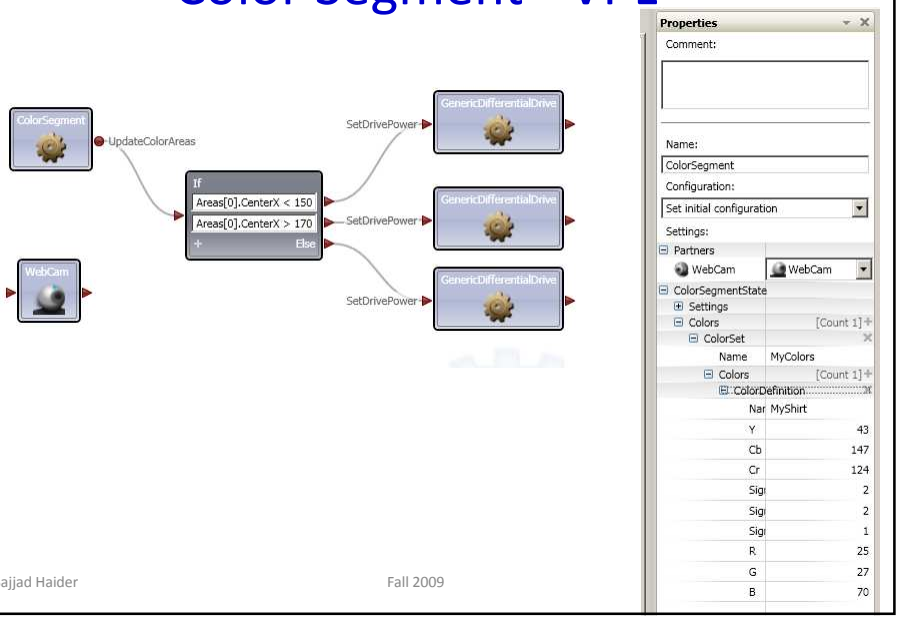
Color Segment Service

- The **Color Segment** service analyzes images from a webcam, or simulated webcam, and based on color definitions declared in the service state, identifies color regions in each image.
- Open “colorsegment.manifest” in DSSMe and run it.
- Type http://localhost:50000/colorsegment in IE.

Color Segment Service

- To define a color, perform the following sequence of actions.
  - Identify a region in the source image that has the color that you want to define.
  - Click on one corner of that region in the source image.
  - Click on another corner of the region. This will cause a red rectangle to be displayed over the image showing the area that will be considered when defining the color.
  - If you are satisfied with the region that has been selected:
    - Click on the name of an existing color definition, or
    - Type the name of a new color definition in the **Name** text box of the **Add Entry** row at the bottom of the list of definitions.
  - Click on the **Add** button to add the new definition.

Color Segment - VPL



Color Segment Service – C#

- To use the **ColorSegment** service from another service you need to add the ColorSegment as a partner to your service. First add a reference to the **ColorSegment** service proxy to your project, then at the top of the service add a using statement.
  - using cs = Microsoft.Robotics.Services.Sample.ColorSegment.Proxy;
- Declare partnership
  - [Partner("ColorSegment", Contract = cs.Contract.Identifier, CreationPolicy = PartnerCreationPolicy.UsePartnerListEntry)]
  - cs.ColorSegmentOperations _colorSegmentPort = new cs.ColorSegmentOperations();
  - cs.ColorSegmentOperations _colorSegmentNotify = new cs.ColorSegmentOperations();

Color Segment Service – C# (Cont'd)

- In the Start method of your class, Subscribe to the **ColorSegment** service, in this example we only subscribe to the cs.UpdateColorAreas message. Also activate a receiver for the notifications from the service.
  - protected overide void Start()

```
{ base.Start();
 _colorSegmentPort.Subscribe(_colorSegmentNotify,
 typeof(cs.UpdateColorAreas));
 Activate( Arbiter.Receive<cd.UpdateColorAreas>(true,
 _colorSegmentNotify, OnUpdateColorAreas) );
}
```

Color Segment Service – C# (Cont'd)

- Then implement the handler method that will be called when the **ColorSegment** service processes an image frame and identifies a color.
  - void OnUpdateColorAreas(cs.UpdateColorAreas update)

```
{
    // process color area information
}
```


Vision in Robotics – Tutorial # 7

- This tutorial shows how to use the Blob Tracker in order to make a follow me service, using a mounted camera in the robot.

Create Partnership

```
[Partner("XInputGamePad", Contract =
    xinput.Contract.Identifier, CreationPolicy =
    PartnerCreationPolicy.CreateAlways)]
    xinput.XInputGamepadOperations _xinputPort = new
    xinput.XInputGamepadOperations();
    xinput.XInputGamepadOperations _xinputNotify = new
    xinput.XInputGamepadOperations();
[Partner("BlobTracker", Contract = blob.Contract.Identifier,
    CreationPolicy = PartnerCreationPolicy.UseExisting)]
    blob.BlobTrackerOperations _blobPort = new
    blob.BlobTrackerOperations(); blob.BlobTrackerOperations
    _blobNotify = new blob.BlobTrackerOperations();
```


Activate Handlers

```
_xinputPort.Subscribe(_xinputNotify);
_blobPort.Subscribe(_blobNotify);

Activate<ITask>(
    Arbiter.Receive<xinput.ButtonsChanged>(true,
    _xinputNotify, OnButtonsChanged),
    Arbiter.Receive<blob.ImageProcessed>(true,
    _blobNotify, OnImageProcessed));
```

Blob Tracker Handler

- The OnImageProcessed method will handle the notification of an image processed by the Blob Tracker and is used for the follow me functionality.
- In this case if the Blob Tracker, tracks an area of more than an X quantity, then the robot will be drive to that object, if not the robot will make turns seeking for the correct tracking.

Blob Tracker Handler (Cont'd)

```
void OnImageProcessed(blob.ImageProcessed
   imageProcessed)
{ if (_followMe)
   { if (imageProcessed.Body.Results.Count == 1)
      { if (imageProcessed.Body.Results[0].Area >
           100)  //object detected
        { _drivePort.SetDrivePower(0.5, 0.5); }
       else //search object
        { _drivePort.SetDrivePower(-0.3, 0.3); }
      }
   }
 }
```

Creating Simulation Environment through Code

- We can create a simulation scene through C# code as well.
- First we need to create partnership with Simulation Engine and then we can use built-in entities to add sky, ground, arena and other obstacles in the simulated environment.
- Robots can also be added in a similar manner.

Create Partnership

- using engine = Microsoft.Robotics.Simulation.Engine.Proxy;
- [Partner("SimulationEngine", Contract = engine.Contract.Identifier, CreationPolicy = PartnerCreationPolicy.UseExistingOrCreate)]

```
engine.SimulationEnginePort _simulationEnginePort
        = new engine.SimulationEnginePort();
engine.SimulationEnginePort _simulationEngineNotify
        = new engine.SimulationEnginePort();
```

Code to Setup Camera

```
protected override void Start()
 {

        base.Start();
        _state.Scale = 1.0f;

        SetupCamera();
        PopulateWorld();
}

private void SetupCamera()
     {
        // Set up initial view
        CameraView view = new CameraView();
        view.EyePosition = new Vector3(-1.65f, 1.63f, -0.29f);
        view.LookAtPoint = new Vector3(-0.68f, 1.38f, -0.18f);
        SimulationEngine.GlobalInstancePort.Update(view);
}
```


Populate the World

```
private void PopulateWorld()
     {
        AddSky();
        AddGround();
        BuildArena();
        SimulationEngine.GlobalInstancePort.Insert(new LegoNXTTribot(new Vector3(2, 0, 0)));
     }

 void AddSky()
     {
        // Add a sky using a static texture. We will use the sky texture
        // to do per pixel lighting on each simulation visual entity
        SkyDomeEntity sky = new SkyDomeEntity("skydome.dds", "sky_diff.dds");
        SimulationEngine.GlobalInstancePort.Insert(sky);

        // Add a directional light to simulate the sun.
        LightSourceEntity sun = new LightSourceEntity();
        sun.State.Name = "Sun";
        sun.Type = LightSourceEntityType.Directional;
        sun.Color = new Vector4(0.8f, 0.8f, 0.8f, 1);
        sun.Direction = new Vector3(0.5f, -.75f, 0.5f);
        SimulationEngine.GlobalInstancePort.Insert(sun);
     }
```

Add Ground

```
void AddGround()
    {
      // Create a slab on which the arena rests
      BoxShape groundShape = new BoxShape(new BoxShapeProperties(
        0, new Pose(), new Vector3(24, 1, 16)));
      SingleShapeEntity ground = new SingleShapeEntity(groundShape, new Vector3(0, 0, 0));
      ground.State.Name = "Ground Slab";
      ground.State.Pose = new Pose(new Vector3(0, -0.5f, 0));
      ground.State.Assets.DefaultTexture = "BrickWall.dds";
      groundShape.State.DiffuseColor = new Vector4(0.1f, 0.1f, 0.1f, 1);
      groundShape.State.Material = new MaterialProperties("ground",
            0.2f, // restitution
            0.5f, // dynamic friction
            0.5f);// static friction
      SimulationEngine.GlobalInstancePort.Insert(ground);
    }
```


Build Arena

```
void BuildArena()
    {
      MaterialProperties bouncyMaterial = new MaterialProperties("Bouncy", 1.0f, 0.5f, 0.6f);
      MaterialProperties SlickMaterial = new MaterialProperties("Slick", 1.0f, 0.0f, 0.0f);
      BoxShape[] boxShapes = new BoxShape[]
      {
        new BoxShape(new BoxShapeProperties(0,
          new Pose(new Vector3(0, 0.5f*_state.Scale, -4*_state.Scale)),  new Vector3(12*_state.Scale, 1*_state.Scale, 0.25f*_state.Scale))),
        new BoxShape(new BoxShapeProperties(0,
          new Pose(new Vector3(0, 0.5f*_state.Scale, 4*_state.Scale)),   new Vector3(12*_state.Scale, 1*_state.Scale, 0.25f*_state.Scale))),
        new BoxShape(new BoxShapeProperties(0,
          new Pose(new Vector3(-(6f - 0.125f)*_state.Scale, 0.5f*_state.Scale, 0)),   new Vector3(0.25f*_state.Scale, 1*_state.Scale, 8f*_state.Scale))),
        new BoxShape(new BoxShapeProperties(0,
          new Pose(new Vector3( (6f - 0.125f)*_state.Scale, 0.5f*_state.Scale, 0)),   new Vector3(0.25f*_state.Scale, 1*_state.Scale, 8f*_state.Scale))),
      };
      boxShapes[0].State.Material = bouncyMaterial;
      boxShapes[1].State.Material = bouncyMaterial;
      boxShapes[2].State.Material = bouncyMaterial;
      boxShapes[3].State.Material = bouncyMaterial;

      MultiShapeEntity arena = new MultiShapeEntity(boxShapes, null);
      arena.State.Name = "BallCourt";
      arena.State.Assets.DefaultTexture = "BrickWall.dds";
      SimulationEngine.GlobalInstancePort.Insert(arena);
    }
```