innoBotService V&R

innoBot Media

innoBody <DIR>

innoBody.bos

innoBody.mtl

innoBody.obj

innoCar <DIR>

Sonor <DIR>

Sonor.bos

Sonor.mtl

Sonor.obj

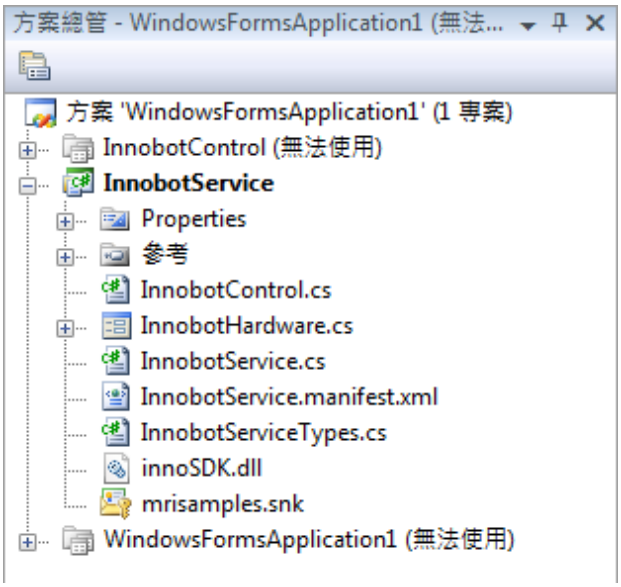
Wheel.bos

Wheel.mtl

Wheel.obj

Install dir: “Microsoft Robotics Dev Studio 2008 R2\store\media”

innoBotService Project Files



innoBotControl.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Runtime.InteropServices;

namespace InnobotService
{
    static public class InnobotControl
    {
        [DllImport("innoSDK.dll", SetLastError = true, CharSet = CharSet.Unicode)]
        internal static extern bool inno_Connect();

        [DllImport("innoSDK.dll", SetLastError = true, CharSet = CharSet.Unicode)]
        internal static extern void inno_Disconnect();

        [DllImport("innoSDK.dll", SetLastError = true, CharSet = CharSet.Unicode)]
        internal static extern bool inno_IsConnect();

        [DllImport("innoSDK.dll", SetLastError = true, CharSet = CharSet.Unicode)]
        internal static extern System.UInt32 inno_ResetTarget();

        //       DWORD PASCAL EXPORT Sonar_SetPosition(WORD a_InPos);
        [DllImport("innoSDK.dll", SetLastError = true, CharSet = CharSet.Unicode)]
        internal static extern System.UInt32 Sonar_SetPosition(System.UInt16 a_InPos);

        //       DWORD PASCAL EXPORT inno_DebugInLong(int a_inLongData);
        [DllImport("innoSDK.dll", SetLastError = true, CharSet = CharSet.Unicode)]
        internal static extern System.UInt32 inno_DebugInLong(System.Int32 a_InSpeed);

        //       DWORD PASCAL EXPORT inno_DebugInFloat(float a_infData);
        [DllImport("innoSDK.dll", SetLastError = true, CharSet = CharSet.Unicode)]
        internal static extern System.UInt32 inno_DebugInFloat(System.Single a_infData);

        //       DWORD PASCAL EXPORT inno_DebugInInteger(short a_inshortData);
        [DllImport("innoSDK.dll", SetLastError = true, CharSet = CharSet.Unicode)]
        internal static extern System.UInt32 inno_DebugInInteger(System.UInt16 a_inshortData);

        //       DWORD PASCAL EXPORT inno_DebugInSByte(char a_insbyteData);
```

```
58          [DllImport("innoSDK.dll", SetLastError = true, CharSet = CharSet.Unicode)]
59          internal static extern System.UInt32 inno_DebugInSByte(System.Char a_insbyteData);
60
61          //       DWORD PASCAL EXPORT inno_DebugInByte(BYTE a_inByteData);
62          [DllImport("innoSDK.dll", SetLastError = true, CharSet = CharSet.Unicode)]
63          internal static extern System.UInt32 inno_DebugInByte(System.Byte a_inByteData);
64
65          //       DWORD PASCAL EXPORT inno_DebugInDWORD(unsigned int a_inDWORDData);
66          [DllImport("innoSDK.dll", SetLastError = true, CharSet = CharSet.Unicode)]
67          internal static extern System.UInt32 inno_DebugInDWORD(System.UInt32 a_inDWORDData);
68
69          //       DWORD PASCAL EXPORT inno_DebugInWORD(unsigned short a_inWORDData);
70          [DllImport("innoSDK.dll", SetLastError = true, CharSet = CharSet.Unicode)]
71          internal static extern System.UInt32 inno_DebugInWORD(System.UInt16 a_inWORDData);
72
73          //       DWORD PASCAL EXPORT inno_DebugOut_SBCFloat(float* a_outfData);
74          [DllImport("innoSDK.dll", SetLastError = true, CharSet = CharSet.Unicode)]
75          internal static extern unsafe System.UInt32 inno_DebugOut_SBCFloat(System.Single[] a_outfData);
76
77          //       DWORD PASCAL EXPORT Motor_SetLeftWheelSpeed(WORD a_InSpeed);
78          [DllImport("innoSDK.dll", SetLastError = true, CharSet = CharSet.Unicode)]
79          internal static extern System.UInt32 Motor_SetLeftWheelSpeed(System.UInt16 a_InSpeed);
80
81          //       DWORD PASCAL EXPORT Motor_SetRightWheelSpeed(WORD a_InSpeed);
82          [DllImport("innoSDK.dll", SetLastError = true, CharSet = CharSet.Unicode)]
83          internal static extern System.UInt32 Motor_SetRightWheelSpeed(System.UInt16 a_InSpeed);
84
85          //       BOOL PASCAL EXPORT Sonar_GetTCRT5000Status(unsigned char* a_OutData);
86          [DllImport("innoSDK.dll", SetLastError = true, CharSet = CharSet.Unicode)]
87          internal static extern unsafe System.Int32 Sonar_GetTCRT5000Status(System.Byte[] a_OutData);
88
89          //       DWORD PASCAL EXPORT Sonar_Ranging(unsigned char a_inChannel);
90          [DllImport("innoSDK.dll", SetLastError = true, CharSet = CharSet.Unicode)]
91          internal static extern System.UInt32 Sonar_Ranging(System.Byte a_inChannel);
92
93          //       DWORD PASCAL EXPORT Sonar_GetDistance(unsigned char a_inChannel, unsigned char* a_poutStatus,
94  unsigned short* a_poutWORD);
95          [DllImport("innoSDK.dll", SetLastError = true, CharSet = CharSet.Unicode)]
96          internal static extern unsafe System.UInt32 Sonar_GetDistance(System.Byte a_inChannel, System.Byte[]
97  a_poutStatus, System.UInt16[] a_poutWORD);
```

```
98
99      }
100 }
```

innoBotServiceTypes.cs

```
101
102 using System;
103 using System.Collections.Generic;
104 using System.ComponentModel;
105 using Microsoft.Ccr.Core;
106 using Microsoft.Dss.Core.Attributes;
107 using Microsoft.Dss.ServiceModel.Dssp;
108 using Microsoft.Dss.ServiceModel.DsspServiceBase;
109 using W3C.Soap;
110
111 #region Simulation Entity
112 using Microsoft.Robotics.Simulation.Engine;
113 using Microsoft.Robotics.Simulation;
114 using Microsoft.Robotics.Simulation.Physics;
115 using Microsoft.Robotics.PhysicalModel;
116 #endregion
117
118 namespace InnobotService
119 {
120     /// <summary>
121     /// InnobotService contract class
122     /// </summary>
123     public sealed class Contract
124     {
125         /// <summary>
126         /// DSS contract identifer for InnobotService
127         /// </summary>
128         [DataMember]
129         public const string Identifier = "http://schemas.tempuri.org/2009/12/innobotservice.html";
130     }
131
132     /// <summary>
133     /// InnobotService state
134     /// </summary>
```

```
135         [DataContract]
136         public class InnobotServiceState
137         {
138         }
139
140         /// <summary>
141         /// InnobotService main operations port
142         /// </summary>
143         [ServicePort]
144         public class InnobotServiceOperations : PortSet<
145             DsspDefaultLookup,
146             DsspDefaultDrop,
147             Get,
148             Subscribe,
149             InnoConnectMessage,
150             InnoIsConnectMessage,
151             InnoResetMessage,
152             InnoDisconnectMessage,
153             InnoSetLeftWheelSpeedMessage,
154             InnoSetRightWheelSpeedMessage,
155             InnoSetSonarPositionMessage,
156             InnoSetWheelsSpeedMessage>
157         {
158         }
159
160         /// <summary>
161         /// InnobotService get operation
162         /// </summary>
163         public class Get : Get<GetRequestType, PortSet<InnobotServiceState, Fault>>
164         {
165             /// <summary>
166             /// Creates a new instance of Get
167             /// </summary>
168             public Get()
169             {
170             }
171
172             /// <summary>
173             /// Creates a new instance of Get
174             /// </summary>
```

```
175         /// <param name="body">the request message body</param>
176         public Get(GetRequestType body)
177             : base(body)
178         {
179         }
180 
181         /// <summary>
182         /// Creates a new instance of Get
183         /// </summary>
184         /// <param name="body">the request message body</param>
185         /// <param name="responsePort">the response port for the request</param>
186         public Get(GetRequestType body, PortSet<InnobotServiceState, Fault> responsePort)
187             : base(body, responsePort)
188         {
189         }
190     }
191 
192     /// <summary>
193     /// InnobotService subscribe operation
194     /// </summary>
195     public class Subscribe : Subscribe<SubscribeRequestType, PortSet<SubscribeResponseType, Fault>>
196     {
197         /// <summary>
198         /// Creates a new instance of Subscribe
199         /// </summary>
200         public Subscribe()
201         {
202         }
203 
204         /// <summary>
205         /// Creates a new instance of Subscribe
206         /// </summary>
207         /// <param name="body">the request message body</param>
208         public Subscribe(SubscribeRequestType body)
209             : base(body)
210         {
211         }
212 
213         /// <summary>
214         /// Creates a new instance of Subscribe
```

```
215         /// </summary>
216         /// <param name="body">the request message body</param>
217         /// <param name="responsePort">the response port for the request</param>
218         public Subscribe(SubscribeRequestType body, PortSet<SubscribeResponseType, Fault> responsePort)
219             : base(body, responsePort)
220         {
221         }
222     }
223
224
225     #region Service Request Messages
226
227     [DataContract]
228     public class InnoConnectRequestType
229     {
230     }
231
232     public class InnoConnectMessage : Upsert<InnoConnectRequestType, PortSet<DefaultUpsertResponseType, Fault>>
233     {
234         public InnoConnectMessage() { }
235
236         public InnoConnectMessage(InnoConnectRequestType body)
237             : base(body)
238         {
239         }
240
241         public InnoConnectMessage(InnoConnectRequestType body, PortSet<DefaultUpsertResponseType, Fault>
242 responsePort)
243             : base(body, responsePort)
244         {
245         }
246     }
247
248     [DataContract]
249     public class InnoIsConnectRequestType
250     {
251     }
252
253     public class InnoIsConnectMessage : Upsert<InnoIsConnectRequestType, PortSet<DefaultUpsertResponseType,
254 Fault>>
```

```
    {
        public InnoIsConnectMessage() { }

        public InnoIsConnectMessage(InnoIsConnectRequestType body)
            : base(body)
        {
        }

        public InnoIsConnectMessage(InnoIsConnectRequestType body, PortSet<DefaultUpsertResponseType, Fault>
responsePort)
            : base(body, responsePort)
        {
        }
    }

    [DataContract]
    public class InnoResetRequestType
    {
    }

    public class InnoResetMessage : Upsert<InnoResetRequestType, PortSet<DefaultUpsertResponseType, Fault>>
    {
        public InnoResetMessage() { }

        public InnoResetMessage(InnoResetRequestType body)
            : base(body)
        {
        }

        public InnoResetMessage(InnoResetRequestType body, PortSet<DefaultUpsertResponseType, Fault> responsePort)
            : base(body, responsePort)
        {
        }
    }

    [DataContract]
    public class InnoDisconnectRequestType
    {
    }

```

```
    public class InnoDisconnectMessage : Upsert<InnoDisconnectRequestType, PortSet<DefaultUpsertResponseType,
Fault>>
    {
        public InnoDisconnectMessage() { }

        public InnoDisconnectMessage(InnoDisconnectRequestType body)
            : base(body)
        {
        }

        public InnoDisconnectMessage(InnoDisconnectRequestType body, PortSet<DefaultUpsertResponseType, Fault>
responsePort)
            : base(body, responsePort)
        {
        }
    }

    [DataContract]
    public class InnoSetLeftWheelSpeedRequestType
    {
        int _speed;

        [DataMember]
        public int Speed
        {
            get { return _speed; }
            set { _speed = value; }
        }
    }

    public class InnoSetLeftWheelSpeedMessage : Upsert<InnoSetLeftWheelSpeedRequestType,
PortSet<DefaultUpsertResponseType, Fault>>
    {
        public InnoSetLeftWheelSpeedMessage() { }

        public InnoSetLeftWheelSpeedMessage(InnoSetLeftWheelSpeedRequestType body)
            : base(body)
        {
        }

```

```
335         public InnoSetLeftWheelSpeedMessage(InnoSetLeftWheelSpeedRequestType body,
336 PortSet<DefaultUpsertResponseType, Fault> responsePort)
337             : base(body, responsePort)
338         {
339         }
340     }
341 
342     [DataContract]
343     public class InnoSetRightWheelSpeedRequestType
344     {
345         int _speed;
346 
347         [DataMember]
348         public int Speed
349         {
350             get { return _speed; }
351             set { _speed = value; }
352         }
353     }
354 
355     public class InnoSetRightWheelSpeedMessage : Upsert<InnoSetRightWheelSpeedRequestType,
356 PortSet<DefaultUpsertResponseType, Fault>>
357     {
358         public InnoSetRightWheelSpeedMessage() { }
359 
360         public InnoSetRightWheelSpeedMessage(InnoSetRightWheelSpeedRequestType body)
361             : base(body)
362         {
363         }
364 
365         public InnoSetRightWheelSpeedMessage(InnoSetRightWheelSpeedRequestType body,
366 PortSet<DefaultUpsertResponseType, Fault> responsePort)
367             : base(body, responsePort)
368         {
369         }
370     }
371 
372     [DataContract]
373     public class InnoSetSonarPositionRequestType
374     {
```

```
375         int _position;
376
377         [DataMember]
378         public int Position
379         {
380             get { return _position; }
381             set { _position = value; }
382         }
383     }
384
385     public class InnoSetSonarPositionMessage : Upsert<InnoSetSonarPositionRequestType,
386 PortSet<DefaultUpsertResponseType, Fault>>
387     {
388         public InnoSetSonarPositionMessage() { }
389
390         public InnoSetSonarPositionMessage(InnoSetSonarPositionRequestType body)
391             : base(body)
392         {
393         }
394
395         public InnoSetSonarPositionMessage(InnoSetSonarPositionRequestType body,
396 PortSet<DefaultUpsertResponseType, Fault> responsePort)
397             : base(body, responsePort)
398         {
399         }
400     }
401
402     [DataContract]
403     public class InnoSetWheelsSpeedRequestType
404     {
405         int _speedRight;
406
407         [DataMember]
408         public int SpeedRight
409         {
410             get { return _speedRight; }
411             set { _speedRight = value; }
412         }
413
414         int _speedLeft;
```

```
415
416         [DataMember]
417         public int SpeedLeft
418         {
419             get { return _speedLeft; }
420             set { _speedLeft = value; }
421         }
422     }
423
424     public class InnoSetWheelsSpeedMessage : Upsert<InnoSetWheelsSpeedRequestType,
425 PortSet<DefaultUpsertResponseType, Fault>>
426     {
427         public InnoSetWheelsSpeedMessage() { }
428
429         public InnoSetWheelsSpeedMessage(InnoSetWheelsSpeedRequestType body)
430             : base(body)
431         {
432         }
433
434         public InnoSetWheelsSpeedMessage(InnoSetWheelsSpeedRequestType body, PortSet<DefaultUpsertResponseType,
435 Fault> responsePort)
436             : base(body, responsePort)
437         {
438         }
439     }
440
441     #endregion
442
443     #region Simulation
444
445     static public class MathTool
446     {
447         static public Quaternion CreateQuaternionFromRPYAngles(float x, float y, float z)
448         {
449             Quaternion q = new Quaternion();
450             double roll = DegreesToRadians(x);
451             double pitch = DegreesToRadians(y);
452             double yaw = DegreesToRadians(z);
453
454             double cyaw, cpitch, croll, syaw, spitch, sroll;
```

```
455             double cyawcpitch, syawspitch, cyawspitch, syawcpitch;
456
457             cyaw = Math.Cos(0.5f * yaw);
458             cpitch = Math.Cos(0.5f * pitch);
459             croll = Math.Cos(0.5f * roll);
460             syaw = Math.Sin(0.5f * yaw);
461
462             spitch = Math.Sin(0.5f * pitch);
463             sroll = Math.Sin(0.5f * roll);
464
465             cyawcpitch = cyaw * cpitch;
466             syawspitch = syaw * spitch;
467             cyawspitch = cyaw * spitch;
468             syawcpitch = syaw * cpitch;
469
470             q.W = (float)(cyawcpitch * croll + syawspitch * sroll);
471             q.X = (float)(cyawcpitch * sroll + syawspitch * croll);
472             q.Y = (float)(cyawspitch * croll + syawcpitch * sroll);
473             q.Z = (float)(syawcpitch * croll + cyawspitch * sroll);
474
475             return q;
476         }
477
478         static public float DegreesToRadians(float deg)
479         {
480             return (float)(deg * Math.PI / 180.0f);
481         }
482
483         static public float RadiansToDegrees(float rad)
484         {
485             return (float)(rad * 180.0f / Math.PI);
486         }
487
488     }
489
490     public class SonarEntityWithJoint : SingleShapeEntity
491     {
492         LaserRangeFinderEntity _lrfEntity;
493
494         public LaserRangeFinderEntity LrfEntity
```

```
        {
            get { return _lrfEntity; }
            set { _lrfEntity = value; }
        }

        private Joint _joint;

        public Joint Joint
        {
            get { return _joint; }
            set { _joint = value; }
        }


        private Joint _customJoint;

        public Joint CustomJoint
        {
            get { return _customJoint; }
            set { _customJoint = value; }
        }

        float _initialJointAngle;

        public float InitialJointAngle
        {
            get { return _initialJointAngle; }
            set { _initialJointAngle = value; }
        }
        float _jointLimitUpper;

        public float JointLimitUpper
        {
            get { return _jointLimitUpper; }
            set { _jointLimitUpper = value; }
        }
        float _jointLimitLower;

        public float JointLimitLower
        {
```

```
535             get { return _jointLimitLower; }
536             set { _jointLimitLower = value; }
537         }
538 
539         public SonarEntityWithJoint(string name, Vector3 position)
540         {
541             this.State.Name = name + "SonarJoint";
542             this.State.Pose.Position = position;
543             this.State.Assets.Mesh = "Sonor.obj";
544             this.MeshRotation = new Vector3(0, 180, 0);
545             this.BoxShape = new BoxShape(new BoxShapeProperties(0.01f, new Pose(new Vector3(0.005f,0.0136f,0f)),
546 new Vector3(0.022f, 0.019f, 0.01f)));
547             #region Sonar
548             const float SONAR_RANGE = 500.0f;
549             _lrfEntity = new LaserRangeFinderEntity();
550             _lrfEntity.State.Name = name + "Sonar";
551             _lrfEntity.LaserBox = new BoxShape(new BoxShapeProperties(0.01f, new Pose(new
552 Vector3(0,0.0135f,-0.0153f)), new Vector3(0.047f, 0.021f, 0.019f)));
553             RaycastProperties raycastProperties = new RaycastProperties();
554             raycastProperties.StartAngle = (float)-120 / 2.0f;
555             raycastProperties.EndAngle = (float)120 / 2.0f;
556             raycastProperties.AngleIncrement = 0.5f;
557             raycastProperties.Range = SONAR_RANGE;
558             raycastProperties.OriginPose = new Pose();
559             _lrfEntity.RaycastProperties = raycastProperties;
560             this.InsertEntity(_lrfEntity);
561             #endregion
562         }
563 
564         public override void Initialize(Microsoft.Xna.Framework.Graphics.GraphicsDevice device, PhysicsEngine
565 physicsEngine)
566         {
567             base.Initialize(device, physicsEngine);
568 
569             // update the parent joint to match our custom joint parameters
570             if (_customJoint != null)
571             {
572                 if ((ParentJoint != null) && (ParentJoint.InternalHandle != (IntPtr)0))
573                     PhysicsEngine.DeleteJoint((PhysicsJoint)ParentJoint);
574 
```

```
                PhysicsEntity.SolverIterationCount = 128;
                Parent.PhysicsEntity.SolverIterationCount = 128;

                _customJoint.State.Connectors[0].Entity = Parent;
                _customJoint.State.Connectors[1].Entity = this;
                ParentJoint = _customJoint;
                PhysicsEngine.InsertJoint((PhysicsJoint)ParentJoint);

((PhysicsJoint)ParentJoint).SetAngularDriveOrientation(MathTool.CreateQuaternionFromRPYAngles(_initialJointAng
le, 0, 0));

            }
        }


    }

    public class InnobotEntity : DifferentialDriveEntity
    {
        IREntity[] _IREntities;

        public IREntity[] IREntities
        {
            get { return _IREntities; }
            set { _IREntities = value; }
        }
        SonarEntityWithJoint _sonarEntity;

        public SonarEntityWithJoint SonarEntity
        {
            get { return _sonarEntity; }
            set { _sonarEntity = value; }
        }

        public InnobotEntity(string name, Vector3 position, float rotation)
            : base()
        {

            MASS = 80;
            CHASSIS_DIMENSIONS = new Vector3(0.11f, 0.08f, 0.15f);
```

```
            CHASSIS_CLEARANCE = 0.1f;
            FRONT_WHEEL_RADIUS = 0.0279f;
            CASTER_WHEEL_RADIUS = 0.0077f; // = CHASSIS_CLEARANCE / 2; // to keep things simple we make caster a
bit bigger
            FRONT_WHEEL_WIDTH = 0.0137f;  //not used
            CASTER_WHEEL_WIDTH = 0.0077f; //not used
            FRONT_AXLE_DEPTH_OFFSET = -0.3f; // distance of the axle from the center of robot

            State.Name = name;
            State.MassDensity.Mass = MASS;
            State.Pose.Position = position;
            State.Pose.Orientation = MathTool.CreateQuaternionFromRPYAngles(0, rotation, 0);
            State.Assets.Mesh = "innoBody.obj";
            this.MeshRotation = new Vector3(0, 180, 0);
            WheelMesh = "Wheel.obj";

            BoxShapeProperties motorBaseDesc = new BoxShapeProperties("chassis", MASS,
                new Pose(new Vector3(0,0.02f,0.03f)), // no offset in the z/depth axis, since again, its center is
the robot center
                CHASSIS_DIMENSIONS);

            motorBaseDesc.Material = new MaterialProperties("high friction", 0.0f, 1.0f, 20.0f);
            motorBaseDesc.Name = "Chassis";
            ChassisShape = new BoxShape(motorBaseDesc);

            // rear wheel is also called the caster
            CASTER_WHEEL_POSITION = new Vector3(0, // center of chassis
                -0.018f, // distance from ground
                0.084f); // all the way at the back of the robot

            // NOTE: right/left is from the perspective of the robot, looking forward

            FRONT_WHEEL_MASS = 0.10f;

            RIGHT_FRONT_WHEEL_POSITION = new Vector3(
                0.053f,// left of center
                0,// distance from ground of axle
                0f); // distance from center, on the z-axis

            LEFT_FRONT_WHEEL_POSITION = new Vector3(
```

```
                -0.053f,// right of center
                0,// distance from ground of axle
                0f); // distance from center, on the z-axis

            MotorTorqueScaling = 20;

            #region IRENtities

            const float DispersionConeAngle = 8.0f;
            const float Samples = 3.0f;
            const float MaximumRange = (30f * 2.54f / 100.0f);


            _IREntities = new IREntity[5];
            const float r = 0.0663f;
            float angleUnit = (float)MathTool.DegreesToRadians(15.8f);
            for (int i = 0; i < 5; i++)
            {
                float angle = (-2 + i ) * angleUnit;
                _IREntities[i] = new IREntity();
                _IREntities[i].State.Name = name + "_IRsensor" + i.ToString();
                _IREntities[i].State.Pose.Position = new Vector3(r * (float)Math.Sin(angle), -0.021f, -r *
(float)Math.Cos(angle) +  0.0103f);//modify this
                _IREntities[i].State.Pose.Orientation = MathTool.CreateQuaternionFromRPYAngles(-90, 0, 0);
                _IREntities[i].LaserBox = new BoxShape(new BoxShapeProperties(0.001f, new Pose(), new Vector3(0.01f,
0.01f, 0.01f)));//modify this
                _IREntities[i].RaycastProperties = new RaycastProperties();
                _IREntities[i].RaycastProperties.StartAngle = -DispersionConeAngle / 2.0f;
                _IREntities[i].RaycastProperties.EndAngle = DispersionConeAngle / 2.0f;
                _IREntities[i].RaycastProperties.AngleIncrement = DispersionConeAngle / (Samples - 1f);
                _IREntities[i].RaycastProperties.Range = MaximumRange;
                _IREntities[i].RaycastProperties.OriginPose = new Pose();

                this.InsertEntity(_IREntities[i]);
            }

            #endregion

            #region Sonar
```

```
            _sonarEntity = new SonarEntityWithJoint(name, new Vector3(0, 0.07f, -0.038f));
            _sonarEntity.State.MassDensity.AngularDamping = 5000;
            _sonarEntity.State.MassDensity.LinearDamping = 5000;
            JointAngularProperties angular = new JointAngularProperties();
            angular.TwistMode = JointDOFMode.Limited;
            angular.Swing1Mode = JointDOFMode.Locked;
            angular.Swing2Mode = JointDOFMode.Locked;
            angular.LowerTwistLimit = new JointLimitProperties(MathTool.DegreesToRadians(-90), 500000, new
SpringProperties(500000, 100000, 0));
            angular.UpperTwistLimit = new JointLimitProperties(MathTool.DegreesToRadians(90), 500000, new
SpringProperties(500000, 100000, 0));
            angular.TwistDrive = new JointDriveProperties(
                JointDriveMode.Position,
                new SpringProperties(500000, 100000, 0),
                1000000);

            EntityJointConnector[] connectors = new EntityJointConnector[2]
                {
                    new EntityJointConnector(null, new Vector3(1,0,0), new Vector3(0,1,0), new Vector3(0, 0.07f,
-0.038f)),
                    new EntityJointConnector(null, new Vector3(1,0,0), new Vector3(0,1,0), new Vector3(0,0,0))
                };
            _sonarEntity.CustomJoint = new Joint();
            _sonarEntity.CustomJoint.State = new JointProperties(angular, connectors);
            _sonarEntity.CustomJoint.State.Name = _sonarEntity.State.Name + "_Joint";
            _sonarEntity.CustomJoint.State.EnableCollisions = false;
            _sonarEntity.InitialJointAngle = 0;
            _sonarEntity.JointLimitLower = MathTool.DegreesToRadians(-90);
            _sonarEntity.JointLimitUpper = MathTool.DegreesToRadians(90);
            this.InsertEntity(_sonarEntity);

            #endregion

        }

        public void SetAngle(float degree)
        {

((PhysicsJoint)this.SonarEntity.ParentJoint).SetAngularDriveOrientation(MathTool.CreateQuaternionFromRPYAngles
(degree, 0, 0));
```

```
        }

        public override void Update(FrameUpdate update)
        {
            base.Update(update);
        }
    }

    #endregion
}
```

innoBotService.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using Microsoft.Ccr.Core;
using Microsoft.Dss.Core.Attributes;
using Microsoft.Dss.ServiceModel.Dssp;
using Microsoft.Dss.ServiceModel.DsspServiceBase;
using W3C.Soap;
using submgr = Microsoft.Dss.Services.SubscriptionManager;
using Microsoft.Ccr.Adapters.WinForms;
using Microsoft.Robotics.PhysicalModel;

#region

using Microsoft.Robotics.Simulation.Engine;
using drive = Microsoft.Robotics.Services.Simulation.Drive.Proxy;
using lrf = Microsoft.Robotics.Services.Simulation.Sensors.Sonar.Proxy;
using ir = Microsoft.Robotics.Services.Simulation.Sensors.Infrared.Proxy;

#endregion

namespace InnobotService
{
    [Contract(Contract.Identifier)]
    [DisplayName("InnobotService")]
    [Description("InnobotService service (no description provided)")]
```

```
772     class InnobotService : DsspServiceBase
773     {
774         /// <summary>
775         /// Service state
776         /// </summary>
777         [ServiceState]
778         InnobotServiceState _state = new InnobotServiceState();
779 
780         /// <summary>
781         /// Main service port
782         /// </summary>
783         [ServicePort("/InnobotService", AllowMultipleInstances = true)]
784         InnobotServiceOperations _mainPort = new InnobotServiceOperations();
785 
786         [SubscriptionManagerPartner]
787         submgr.SubscriptionManagerPort _submgrPort = new submgr.SubscriptionManagerPort();
788 
789 
790         InnobotHardwareEventsPort _innobotEventPorts = new InnobotHardwareEventsPort();
791         InnobotHardware _innobotForm;
792         InnobotEntity _entity;
793         Port<DateTime> _SIMSensorTimerPort = new Port<DateTime>();
794         /// <summary>
795         /// Service constructor
796         /// </summary>
797         public InnobotService(DsspServiceCreationPort creationPort)
798             : base(creationPort)
799         {
800         }
801 
802         /// <summary>
803         /// Service start
804         /// </summary>
805         protected override void Start()
806         {
807 
808             Activate(Arbiter.Interleave(
809                 new TeardownReceiverGroup
810                 (
811 
```

```
                ),
                new ExclusiveReceiverGroup
                (

                ),
                new ConcurrentReceiverGroup
                (
                    Arbiter.ReceiveWithIterator<Event_Connect>(true, _innobotEventPorts, ConnectHandler),
                    Arbiter.ReceiveWithIterator<Event_IsConnect>(true, _innobotEventPorts, IsConnectHandler),
                    Arbiter.ReceiveWithIterator<Event_Reset>(true, _innobotEventPorts, ResetHandler),
                    Arbiter.ReceiveWithIterator<Event_Disconnect>(true, _innobotEventPorts, DisconnectHandler),
                    Arbiter.ReceiveWithIterator<Event_SetLeftWheelSpeed>(true, _innobotEventPorts,
SetLeftWheelSpeedHandler),
                    Arbiter.ReceiveWithIterator<Event_SetRightWheelSpeed>(true, _innobotEventPorts,
SetRightWheelSpeedHandler),
                    Arbiter.ReceiveWithIterator<Event_SetSonarPosition>(true, _innobotEventPorts,
SetSonarPositionHandler),
                    Arbiter.ReceiveWithIterator<SIMEvent_InsertEntity>(true, _innobotEventPorts,
InsertEntityHandler),
                    Arbiter.ReceiveWithIterator<SIMEvent_SetSonarAngle>(true, _innobotEventPorts,
SIMSetSonarAngleHandler)
                )
            ));

            base.Start();

            _innobotForm = new InnobotHardware(_innobotEventPorts);
            WinFormsServicePort.Post(new RunForm(CreateForm));


        }

        System.Windows.Forms.Form CreateForm()
        {
            return _innobotForm;
        }

        #region Server Handler
        /// <summary>
        /// Handles Subscribe messages
```

```
852         /// </summary>
853         /// <param name="subscribe">the subscribe request</param>
854         [ServiceHandler]
855         public void SubscribeHandler(Subscribe subscribe)
856         {
857             SubscribeHelper(_submgrPort, subscribe.Body, subscribe.ResponsePort);
858         }
859
860         [ServiceHandler]
861         public void InnoConnectHandler(InnoConnectMessage connect)
862         {
863             _innobotEventPorts.Post(new Event_Connect());
864             connect.ResponsePort.Post(new DefaultUpsertResponseType());
865         }
866
867         [ServiceHandler]
868         public void InnoIsConnectHandler(InnoIsConnectMessage isconnect)
869         {
870             _innobotEventPorts.Post(new Event_IsConnect());
871         }
872
873         [ServiceHandler]
874         public void InnoResetHandler(InnoResetMessage reset)
875         {
876             _innobotEventPorts.Post(new Event_Reset());
877         }
878
879         [ServiceHandler]
880         public void InnoDisconnectHandler(InnoDisconnectMessage disc)
881         {
882             _innobotEventPorts.Post(new Event_Disconnect());
883         }
884
885         [ServiceHandler]
886         public void InnoSetLeftWheelHandler(InnoSetLeftWheelSpeedMessage set)
887         {
888             _innobotEventPorts.Post(new Event_SetLeftWheelSpeed(set.Body.Speed));
889         }
890
891         [ServiceHandler]
```

```
892         public void InnoSetRightWheelHandler(InnoSetRightWheelSpeedMessage set)
893         {
894             _innobotEventPorts.Post(new Event_SetRightWheelSpeed(set.Body.Speed));
895         }
896
897         [ServiceHandler]
898         public void InnoSetSonarPositionHandler(InnoSetSonarPositionMessage set)
899         {
900             _innobotEventPorts.Post(new Event_SetSonarPosition(set.Body.Position));
901         }
902
903         [ServiceHandler]
904         public void InnoSetWheelsHandler(InnoSetWheelsSpeedMessage set)
905         {
906             _innobotEventPorts.Post(new Event_SetRightWheelSpeed(set.Body.SpeedRight));
907             _innobotEventPorts.Post(new Event_SetLeftWheelSpeed(set.Body.SpeedLeft));
908
909         }
910
911
912         #endregion
913         #region Innobot Event Handlers
914
915         IEnumerator<ITask> ConnectHandler(Event_Connect connect)
916         {
917             if (_innobotForm != null)
918             {
919                 WinFormsServicePort.FormInvoke(
920                     delegate()
921                     {
922                         InnobotControl.inno_Connect();
923                         _innobotForm.UpdateMessage(InnobotControl.inno_IsConnect().ToString());
924                         InnobotControl.inno_ResetTarget();
925                         _innobotForm.UpdateMessage("Connect and Reset innoBot");
926                     }
927                 );
928             }
929
930             yield break;
931         }
```

```
932
933         IEnumerator<ITask> IsConnectHandler(Event_IsConnect isconnect)
934         {
935             if (_innobotForm != null)
936             {
937                 WinFormsServicePort.FormInvoke(
938                     delegate()
939                     {
940                         InnobotControl.inno_IsConnect();
941                         _innobotForm.UpdateMessage("IsConnect Status: " +
942 InnobotControl.inno_IsConnect().ToString());
943                     }
944                 );
945             }
946 
947             yield break;
948         }
949 
950         IEnumerator<ITask> ResetHandler(Event_Reset reset)
951         {
952             if (_innobotForm != null)
953             {
954                 WinFormsServicePort.FormInvoke(
955                     delegate()
956                     {
957                         InnobotControl.inno_ResetTarget();
958                         _innobotForm.UpdateMessage("Reset innoBot");
959                     }
960                 );
961             }
962 
963             yield break;
964         }
965 
966         IEnumerator<ITask> DisconnectHandler(Event_Disconnect dis)
967         {
968             if (_innobotForm != null)
969             {
970                 WinFormsServicePort.FormInvoke(
971                     delegate()
```

```
                    {
                        if (InnobotControl.inno_IsConnect() == true)
                        {
                            InnobotControl.inno_Disconnect();
                            _innobotForm.UpdateMessage("Disconnect innoBot");
                        }
                        else
                        {
                            _innobotForm.UpdateMessage("Already disconnect innoBot");
                        }
                    }
                );
            }

            yield break;
        }

        IEnumerator<ITask> SetLeftWheelSpeedHandler(Event_SetLeftWheelSpeed speed)
        {
            if (_innobotForm != null)
            {
                WinFormsServicePort.FormInvoke(
                    delegate()
                    {

                        _innobotForm.UpdateMessage("Your string is : " +
InnobotControl.Motor_SetLeftWheelSpeed((UInt16)speed.Speed).ToString());
                    }
                );
            }
            yield break;
        }


        IEnumerator<ITask> SetRightWheelSpeedHandler(Event_SetRightWheelSpeed speed)
        {
            if (_innobotForm != null)
            {
                WinFormsServicePort.FormInvoke(
                    delegate()
```

```
                    {

                        _innobotForm.UpdateMessage("Set Right Wheel Speed:" +
InnobotControl.Motor_SetRightWheelSpeed((UInt16)speed.Speed).ToString());
                    }
                );
            }
            yield break;
        }

        IEnumerator<ITask> SetSonarPositionHandler(Event_SetSonarPosition position)
        {
            if (_innobotForm != null)
            {
                WinFormsServicePort.FormInvoke(
                    delegate()
                    {
                        InnobotControl.Sonar_SetPosition((UInt16)position.Position);
                        _innobotForm.UpdateMessage("Set Sonar Position:" +
InnobotControl.Sonar_SetPosition((UInt16)position.Position).ToString());
                    }
                );
            }
            yield break;
        }


        IEnumerator<ITask> InsertEntityHandler(SIMEvent_InsertEntity insert)
        {
            _entity = new InnobotEntity(insert.Name, new Vector3(insert.X,insert.Y,insert.Z), 0);
            SimulationEngine.GlobalInstancePort.Insert(_entity);
            /*Create drive service*/
            drive.Contract.CreateService(ConstructorPort, "http://localhost/" + _entity.State.Name,
                Microsoft.Robotics.Simulation.Partners.CreateEntityPartner(
                    "http://localhost/" + _entity.State.Name));
            /*Create sonar service*/
            lrf.Contract.CreateService(ConstructorPort, "http://localhost/" +
_entity.SonarEntity.LrfEntity.State.Name,
                Microsoft.Robotics.Simulation.Partners.CreateEntityPartner(
                    "http://localhost/" + _entity.SonarEntity.LrfEntity.State.Name));
```

```
            /*Create ir service*/
            for (int i = 0; i < 5; i++)
            {
                ir.Contract.CreateService(ConstructorPort, "http://localhost/" +
_entity.IREntities[i].State.Name,
                Microsoft.Robotics.Simulation.Partners.CreateEntityPartner(
                    "http://localhost/" + _entity.IREntities[i].State.Name));
            }
            /*Create timer port*/
            _SIMSensorTimerPort.Post(DateTime.Now);
            Activate(Arbiter.Receive(true, _SIMSensorTimerPort, SIMSensorTimerHandler));

            if (_innobotForm != null)
            {
                WinFormsServicePort.FormInvoke(
                    delegate()
                    {
                        _innobotForm.UpdateMessage("Insert Entity");
                        _innobotForm.SetMaxDistance(0.05f,
_entity.SonarEntity.LrfEntity.RaycastProperties.Range);
                    }
                );
            }

            yield break;
        }

        void SIMSensorTimerHandler(DateTime signal)
        {
            WinFormsServicePort.FormInvoke(
                    delegate()
                    {
                        _innobotForm.UpdateSIMSensors(_entity.IREntities[0].Distance,
                            _entity.IREntities[1].Distance,
                            _entity.IREntities[2].Distance,
                            _entity.IREntities[3].Distance,
                            _entity.IREntities[4].Distance,
                            0);
                    }
                );
```

```
1092                // Set the timer for the next tick
1093                Activate(
1094                    Arbiter.Receive(false, TimeoutPort(1000),
1095                        delegate(DateTime time)
1096                        {
1097                            _SIMSensorTimerPort.Post(time);
1098                        }
1099                    )
1100                );
1101            }
1102
1103            IEnumerator<ITask> SIMSetSonarAngleHandler(SIMEvent_SetSonarAngle angle)
1104            {
1105                _entity.SetAngle(angle.Degree);
1106                if (_innobotForm != null)
1107                {
1108                    WinFormsServicePort.FormInvoke(
1109                        delegate()
1110                        {
1111                            _innobotForm.UpdateMessage("Set Sonar Angle");
1112                        }
1113                    );
1114                }
1115                yield break;
1116            }
1117
1118            #endregion
1119        }
1120    }
```

InnobotHardware

InnobotHardware.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;

using System.Text;
using System.Windows.Forms;
using Microsoft.Ccr.Core;

namespace InnobotService
{
    partial class InnobotHardware : Form
    {
        InnobotHardwareEventsPort _eventsPort;

        public InnobotHardware(InnobotHardwareEventsPort port)
        {
            _eventsPort = port;
            InitializeComponent();
```

```
1144        }
1145
1146        public void UpdateMessage(string message)
1147        {
1148            label_message.Text = message;
1149        }
1150
1151        private void button_connect_Click(object sender, EventArgs e)
1152        {
1153            _eventsPort.Post(new Event_Connect());
1154        }
1155
1156        private void button_isconnect_Click(object sender, EventArgs e)
1157        {
1158            _eventsPort.Post(new Event_IsConnect());
1159        }
1160
1161        private void button_reset_Click(object sender, EventArgs e)
1162        {
1163            _eventsPort.Post(new Event_Reset());
1164        }
1165
1166        private void button_disconnect_Click(object sender, EventArgs e)
1167        {
1168            _eventsPort.Post(new Event_Disconnect());
1169        }
1170
1171        private void button_setLeftWheelSpeed_Click(object sender, EventArgs e)
1172        {
1173            _eventsPort.Post(new Event_SetLeftWheelSpeed(Convert.ToInt16(textBox_setLSpeed.Text)));
1174        }
1175
1176        private void button_setRightWheelSpeed_Click(object sender, EventArgs e)
1177        {
1178            _eventsPort.Post(new Event_SetRightWheelSpeed(Convert.ToInt16(textBox_setLSpeed.Text)));
1179        }
1180
1181        private void button1_Click(object sender, EventArgs e)
1182        {
1183            _eventsPort.Post(new SIMEvent_InsertEntity()
```

```
            {
                Name = textBox_entityName.Text,
                X = Convert.ToSingle(textBox_SIMX.Text),
                Y = Convert.ToSingle(textBox_SIMY.Text),
                Z = Convert.ToSingle(textBox_SIMZ.Text)
            });
        }

        private void button_SetAngle_Click(object sender, EventArgs e)
        {
            _eventsPort.Post(new SIMEvent_SetSonarAngle(Convert.ToSingle(textBox_SonarJoint.Text)));
        }

        public void UpdateSIMSensors(float ir0, float ir1, float ir2, float ir3, float ir4, float sonar)
        {
            int value0 =(int)((ir0 / _maxIRDistance) * 100f);
            if(value0 >100)
                value0= 100;
            int value1 =(int)((ir1 / _maxIRDistance) * 100f);
            if(value1 >100)
                value1= 100;
            int value2 =(int)((ir2 / _maxIRDistance) * 100f);
            if(value2 >100)
                value2= 100;
            int value3 =(int)((ir3 / _maxIRDistance) * 100f);
            if(value3 >100)
                value3= 100;
            int value4 =(int)((ir4 / _maxIRDistance) * 100f);
            if(value4 >100)
                value4= 100;
            int valueSonar = (int)((sonar / _maxSonarDistance) * 100f);
            if(valueSonar > 100)
                valueSonar = 100;
            progressBar_SIMIR0.Value = value0;
            progressBar_SIMIR1.Value = value1;
            progressBar_SIMIR2.Value = value2;
            progressBar_SIMIR3.Value = value3;
            progressBar_SIMIR4.Value = value4;
            progressBar_Sonar.Value = valueSonar;
            label_SIMIR0.Text = "IR Sensor0:" + (ir0 * 100f).ToString() + "cm";
```

```
            label_SIMIR1.Text = "IR Sensor1:" + (ir1 * 100f).ToString() + "cm";
            label_SIMIR2.Text = "IR Sensor2:" + (ir2 * 100f).ToString() + "cm";
            label_SIMIR3.Text = "IR Sensor3:" + (ir3 * 100f).ToString() + "cm";
            label_SIMIR4.Text = "IR Sensor4:" + (ir4 * 100f).ToString() + "cm";
            label_SIMsonar.Text = "Sonar:" + (sonar * 100f).ToString() + "cm";

        }

        public void SetMaxDistance(float ir, float sonar)
        {
            _maxIRDistance = ir;
            _maxSonarDistance = sonar;
        }

        float _maxIRDistance;
        float _maxSonarDistance;

        private void button_SetSonarPosition_Click(object sender, EventArgs e)
        {
            _eventsPort.Post(new Event_SetSonarPosition(Convert.ToInt32(textBox_SetSonarPosition.Text)));
        }

    }

    class InnobotHardwareEventsPort: PortSet<
        Event_Connect,
        Event_IsConnect,
        Event_Reset,
        Event_Disconnect,
        Event_SetLeftWheelSpeed,
        Event_SetRightWheelSpeed,
        Event_SetSonarPosition,
        SIMEvent_InsertEntity,
//        SIMEvent_SetLeftWheelSpeed,
//        SIMEvent_SetRightWheelSpeed,
        SIMEvent_SetSonarAngle
        >{}

    class Event_Connect
    {
```

```
1264            public Event_Connect() { }
1265        }
1266
1267        class Event_IsConnect
1268        {
1269            public Event_IsConnect() { }
1270        }
1271
1272        class Event_Reset
1273        {
1274            public Event_Reset() { }
1275        }
1276
1277        class Event_Disconnect
1278        {
1279            public Event_Disconnect() { }
1280        }
1281
1282        class Event_SetLeftWheelSpeed
1283        {
1284            private int _speed;
1285
1286            public int Speed
1287            {
1288                get { return _speed; }
1289            }
1290
1291            public Event_SetLeftWheelSpeed(int speed)
1292            {
1293                _speed = speed;
1294            }
1295        }
1296
1297        class Event_SetRightWheelSpeed
1298        {
1299            private int _speed;
1300
1301            public int Speed
1302            {
1303                get { return _speed; }
```

```
1304            }
1305
1306            public Event_SetRightWheelSpeed(int speed)
1307            {
1308                _speed = speed;
1309            }
1310        }
1311
1312        class Event_SetSonarPosition
1313        {
1314            private int _position;
1315
1316            public int Position
1317            {
1318                get { return _position; }
1319                set { _position = value; }
1320            }
1321
1322            public Event_SetSonarPosition(int position)
1323            {
1324                _position = position;
1325            }
1326        }
1327
1328
1329        class SIMEvent_InsertEntity
1330        {
1331            float _x;
1332
1333            public float X
1334            {
1335                get { return _x; }
1336                set { _x = value; }
1337            }
1338
1339            float _y;
1340
1341            public float Y
1342            {
1343                get { return _y; }
```

```
            set { _y = value; }
        }

        float _z;

        public float Z
        {
            get { return _z; }
            set { _z = value; }
        }

        string _name;

        public string Name
        {
            get { return _name; }
            set { _name = value; }
        }

        public SIMEvent_InsertEntity() { }
    }

    class SIMEvent_SetSonarAngle
    {
        private float _degree;

        public float Degree
        {
            get { return _degree; }
            set { _degree = value; }
        }

        public SIMEvent_SetSonarAngle(float degree)
        {
            _degree = degree;
        }
    }
}
```