11 Appendix: Source Code

Master 4WD/4WS vehicle control algorithm v1 - Variables.nb

This file contains 2D representation of vehicle control with SpaceMouse. In this construction, all dimensions will be parameterized, that is, everything will be done with with variables so that they can be changed any time.

phy prefix is for physical dimensions

```
In[•]:= phyCarHalfWidth = 1; phyCarHalfLength = 2; phyGapBetweenCarAndWheel = 0.8;
phyWheelX = phyCarHalfWidth + phyGapBetweenCarAndWheel;
phyWheely = 0.8 phyCarHalfLength;
phyWheelHalfThickness = 0.2;
phyWheelRadius = 0.5;
phyDrivingFieldX = phyDrivingFieldY = 10;
phyDrivingFieldGridLineGap = 1;
phyWheelFrontRight = {phyWheelX, phyWheely};
phyWheelBackRight = {phyWheelX, -phyWheely};
phyWheelFrontLeft = {-phyWheelX, +phyWheely};
phyWheelBackLeft = {-phyWheelX, -phyWheely};
```

Current moving directions relative to its vehicle's coordinate system, i.e., from the driver's point of view. Directions are specified with unit vectors.

```
In[•]:= dirBody = {0, 1};
```

These are variable for the current positions

```
In[•]:= posCurveCenter = {-1000, 0};
posBody = {0, 0};
posWheelFrontRight := RotationTransform[rotBody, {0, 0}][phyWheelFrontRight];
posWheelBackRight := RotationTransform[rotBody, {0, 0}][phyWheelBackRight];
posWheelFrontLeft := RotationTransform[rotBody, {0, 0}][phyWheelFrontLeft];
posWheelBackLeft := RotationTransform[rotBody, {0, 0}][phyWheelBackLeft];
```

Printed by Wolfram Mathematica Student Edition

Rotations. Global angle in the frame of reference of the background grid. This is not needed in actual driving but is needed to show the motion correctly. Rotations are specified with angles in radian

```
In[•]:= rotBody = 0;
```

col prefix is for colors

```
In[•]:= colCarBody = White; colCarEdge = Gray;
colWheelBody = White; colWheelEdge = Black;
colVelocityArrow = Red; colCurve = Green;  colGridLines = LightGray;
```

parameters These are each wheel's angle relative to the main body, and each wheel's speed to be sent to the motors.

```
In[•]:= angleFrontRight = 0; angleBackRight = 0; angleFrontLeft = 0; angleBackLeft = 0;
speedFrontRight = 0; speedBackRight = 0; speedFrontLeft = 0; speedBackLeft = 0;
```

Printed by Wolfram Mathematica Student Edition

combined commands
pos is Position

In[•]:=
```
comBackground =
  {
   GridLines -> {Table[i,
      {i, -phyDrivingFieldX, phyDrivingFieldX, phyDrivingFieldGridLineGap}], Table[
      i, {i, -phyDrivingFieldY, phyDrivingFieldY, phyDrivingFieldGridLineGap}]},
   GridLinesStyle → Directive[colGridLines],
   PlotRange → {phyDrivingFieldX {-1, 1}, phyDrivingFieldY {-1, 1}},
   ImageSize → 800
  };
Clear[comCar]

comCar[posBody_, dirBodyAngle_] := Module[{},

   {EdgeForm[colCarEdge], colCarBody,

    Rotate[

     (*Rectangle[-{phyCarHalfWidth,phyCarHalfLength}+posBody,
      {phyCarHalfWidth,phyCarHalfLength}+posBody],*)
     Polygon[# + posBody & /@ {{-phyCarHalfWidth, -phyCarHalfLength},
        {-phyCarHalfWidth, phyCarHalfLength},
        {0, 1.4 phyCarHalfLength}, {phyCarHalfWidth, phyCarHalfLength},
        {phyCarHalfWidth, -phyCarHalfLength}}],
     rotBody, posBody]

    }
  ];
```

```
In[•]:= Clear[comWheels]
comWheels[posBody_, dirBodyAngle_, posCurveCenter_] :=
 {
  dirFrontRight = ArcTan @@ (posCurveCenter - posWheelFrontRight) + If[z2 > 0, Pi, 0];
  dirBackRight = ArcTan @@ (posCurveCenter - posWheelBackRight) + If[z2 > 0, Pi, 0];
  dirFrontLeft = ArcTan @@ (posCurveCenter - posWheelFrontLeft) + If[z2 > 0, Pi, 0];
  dirBackLeft = ArcTan @@ (posCurveCenter - posWheelBackLeft) + If[z2 > 0, Pi, 0];
  (* common to all wheels *)
  EdgeForm[colWheelEdge], colWheelBody,

  (*front right*)

  EdgeForm[Red],
  Rotate[Rectangle[
    posWheelFrontRight + {phyWheelHalfThickness, -phyWheelRadius} + posBody,
    posWheelFrontRight + {-phyWheelHalfThickness, phyWheelRadius} + posBody],
   dirFrontRight, posWheelFrontRight + posBody],

  (*back right*)
  EdgeForm[Blue],
  Rotate[
   Rectangle[posWheelBackRight + {phyWheelHalfThickness, -phyWheelRadius} + posBody,
    posWheelBackRight + {-phyWheelHalfThickness, phyWheelRadius} + posBody],
   dirBackRight, posWheelBackRight + posBody],
  (*front left*)
  EdgeForm[Purple],
  Rotate[
   Rectangle[posWheelFrontLeft + {phyWheelHalfThickness, -phyWheelRadius} + posBody,
    posWheelFrontLeft + {-phyWheelHalfThickness, phyWheelRadius} + posBody],
   dirFrontLeft, posWheelFrontLeft + posBody],

  (*back left*)
  EdgeForm[Darker[Green]],
  Rotate[
   Rectangle[posWheelBackLeft + {phyWheelHalfThickness, -phyWheelRadius} + posBody,
    posWheelBackLeft + {-phyWheelHalfThickness, phyWheelRadius} + posBody],
   dirBackLeft, posWheelBackLeft + posBody],

  PointSize → Large, Red,
  Point[posWheelFrontRight + posBody], Point[posWheelBackRight + posBody],
  Point[posWheelFrontLeft + posBody], Point[posWheelBackLeft + posBody]

 }
```

```
In[•]:= Clear[comDrawCircles]
comDrawCircles[posBody_, dirBodyAngle_, posCurveCenter_] :=
 {
  (*Circle that passes through the center of the vehicle*)
  colCurve,
  {PointSize → 0.05, Point[posCurveCenter + posBody]},

  bodyRadius = Norm[posCurveCenter];
  wheelFrontRightRadius = Norm[posCurveCenter - posWheelFrontRight];
  wheelBackRightRadius = Norm[posCurveCenter - posWheelBackRight];
  wheelFrontLeftRadius = Norm[posCurveCenter - posWheelFrontLeft];
  wheelBackLeftRadius = Norm[posCurveCenter - posWheelBackLeft];

  Circle[posCurveCenter + posBody, bodyRadius],
  (*front right wheel's path circle*)
  Circle[posCurveCenter + posBody, wheelFrontRightRadius],

  (*back right wheel's path circle*)
  Circle[posCurveCenter + posBody, wheelBackRightRadius],

  (*front left wheel's path circle*)
  Circle[posCurveCenter + posBody, wheelFrontLeftRadius],
  (*back left wheel's path circle*)
  Circle[posCurveCenter + posBody, wheelBackLeftRadius],

  arcLength = 2;
  arrowLength = 0.1;
  Arrowheads[0.02],

  Blue,
  bodyRadius = Norm[posCurveCenter];
  angle = ArcTan @@ (posCurveCenter) ;
  angleEnd = angle + arcLength Sign[z2] / radius;
  direction = -arrowLength Sign[z2] {-Sin[angleEnd], Cos[angleEnd]};
  Circle[posCurveCenter + posBody, bodyRadius, {angle, angleEnd} + Pi],
  arcTip =
   posCurveCenter + bodyRadius {Cos[angleEnd + Pi], Sin[angleEnd + Pi]} + posBody;
  Arrow[{arcTip, arcTip + direction}],

  (*front right wheel direction*)
  Red,
  If[b1 || b2,
   If[b1, reductionFactor = 2000, reductionFactor = 2000], reductionFactor = 1];
```

```
 posFrontRightWheelCenter = (posWheelFrontRight + posBody);
 radius = Norm[posCurveCenter - posWheelFrontRight];
 angle = ArcTan @@ (posCurveCenter - posWheelFrontRight) ;
 angleEnd = angle + arcLength / reductionFactor Sign[z2] / bodyRadius;
 direction = - arrowLength Sign[z2] {- Sin[angleEnd], Cos[angleEnd]};
 Circle[posCurveCenter + posBody, radius, {angle, angleEnd} + Pi],
 arcTip = posCurveCenter + radius {Cos[angleEnd + Pi], Sin[angleEnd + Pi]} + posBody;
 Arrow[{arcTip, arcTip + direction}],

 (*back right wheel direction*)

 posBackRightWheelCenter = (posWheelBackRight + posBody);
 radius = Norm[posCurveCenter - posWheelBackRight];
 angle = ArcTan @@ (posCurveCenter - posWheelBackRight) ;
 angleEnd = angle + arcLength / reductionFactor Sign[z2] / bodyRadius;
 direction = - arrowLength Sign[z2] {- Sin[angleEnd], Cos[angleEnd]};
 Circle[posCurveCenter + posBody, radius, {angle, angleEnd} + Pi],
 arcTip = posCurveCenter + radius {Cos[angleEnd + Pi], Sin[angleEnd + Pi]} + posBody;
 Arrow[{arcTip, arcTip + direction}],

 (*front left wheel direction*)
 posFrontLeftWheelCenter = (posWheelFrontLeft + posBody);
 radius = Norm[posCurveCenter - posWheelFrontLeft];
 angle = ArcTan @@ (posCurveCenter - posWheelFrontLeft) ;
 angleEnd = angle + arcLength / reductionFactor Sign[z2] / bodyRadius;
 direction = - arrowLength Sign[z2] {- Sin[angleEnd], Cos[angleEnd]};
 Circle[posCurveCenter + posBody, radius, {angle, angleEnd} + Pi],
 arcTip = posCurveCenter + radius {Cos[angleEnd + Pi], Sin[angleEnd + Pi]} + posBody;
 Arrow[{arcTip, arcTip + direction}],

 (*back left wheel direction*)
 posBackLeftWheelCenter = (posWheelBackLeft + posBody);
 radius = Norm[posCurveCenter - posWheelBackLeft];
 angle = ArcTan @@ (posCurveCenter - posWheelBackLeft) ;
 angleEnd = angle + arcLength / reductionFactor Sign[z2] / bodyRadius;
 direction = - arrowLength Sign[z2] {- Sin[angleEnd], Cos[angleEnd]};
 Circle[posCurveCenter + posBody, radius, {angle, angleEnd} + Pi],
 arcTip = posCurveCenter + radius {Cos[angleEnd + Pi], Sin[angleEnd + Pi]} + posBody;
 Arrow[{arcTip, arcTip + direction}]
}
```

```
In[•]:= Clear[comWheelLabels]
comWheelLabels[posBody_, dirBodyAngle_] :=
 {
  (* common to all wheels *)
  Arrowheads[0.02],
  (*front right*)
  LightGray,
  Arrow[{1.1 phyWheelFrontRight + posBody, 1.9 phyWheelFrontRight + posBody}],
  (*back right*)
  Arrow[{1.1 phyWheelBackRight + posBody, 1.9 phyWheelBackRight + posBody}],
  (*front left*)
  Arrow[{1.1 phyWheelFrontLeft + posBody, 1.9 phyWheelFrontLeft + posBody}],
  (*back left*)
  Arrow[{1.1 phyWheelBackLeft + posBody, 1.9 phyWheelBackLeft + posBody}],

  Gray,
  Text[Style["Stuff", 18], 2 phyWheelFrontRight + posBody],
  Text[Style["Stuff", 18], 2 phyWheelBackRight + posBody],
  Text[Style["Stuff", 18], 2 phyWheelFrontLeft + posBody],
  Text[Style["Stuff", 18], 2 phyWheelBackLeft + posBody]
 }
```

This will show the calculation for each wheel's speed and angle relative to the main body

```
In[•]:= Clear[comDrawCalculations]
comDrawCalculations[posBody_, dirBodyAngle_, posCurveCenter_] :=
 {


  (* First draw where the direction 0 would be for each wheel *)
  Arrowheads[0.02],
  (* front right wheel *)

  angleFrontRight = (dirFrontRight - rotBody);
  angleFrontRight = Mod[angleFrontRight + Pi, 2 Pi] - Pi;
  angleBackRight = (dirBackRight - rotBody);
  angleBackRight = Mod[angleBackRight + Pi, 2 Pi] - Pi;
  angleFrontLeft = (dirFrontLeft - rotBody);
  angleFrontLeft = Mod[angleFrontLeft + Pi, 2 Pi] - Pi;
  angleBackLeft = (dirBackLeft - rotBody);
  angleBackLeft = Mod[angleBackLeft + Pi, 2 Pi] - Pi;
  speedFrontRight = wheelFrontRightRadius / bodyRadius;
```

```
speedBackRight = wheelBackRightRadius / bodyRadius;
speedFrontLeft = wheelFrontLeftRadius / bodyRadius;
speedBackLeft = wheelBackLeftRadius / bodyRadius;

Gray,
Rotate[Arrow[
  {posWheelFrontRight + posBody, posWheelFrontRight + posBody + {0, arcLength}}],
 rotBody, posWheelFrontRight + posBody],
Rotate[Arrow[{posWheelBackRight + posBody, posWheelBackRight +
    posBody + {0, arcLength}}], rotBody, posWheelBackRight + posBody],
Rotate[Arrow[{posWheelFrontLeft + posBody, posWheelFrontLeft + posBody +
    {0, arcLength}}], rotBody, posWheelFrontLeft + posBody],
Rotate[Arrow[{posWheelBackLeft + posBody, posWheelBackLeft + posBody +
    {0, arcLength}}], rotBody, posWheelBackLeft + posBody],
Pink,
Rotate[Arrow[
  {posWheelFrontRight + posBody, posWheelFrontRight + posBody + {0, arcLength}}],
 rotBody + angleFrontRight, posWheelFrontRight + posBody],
Rotate[Arrow[{posWheelBackRight + posBody,
   posWheelBackRight + posBody + {0, arcLength}}],
 rotBody + angleBackRight, posWheelBackRight + posBody],
Rotate[Arrow[{posWheelFrontLeft + posBody,
   posWheelFrontLeft + posBody + {0, arcLength}}],
 rotBody + angleFrontLeft, posWheelFrontLeft + posBody],
Rotate[Arrow[{posWheelBackLeft + posBody,
   posWheelBackLeft + posBody + {0, arcLength}}],
 rotBody + angleBackLeft, posWheelBackLeft + posBody],

circleRadius = 2;
Rotate[Circle[posWheelFrontRight + posBody, circleRadius,
  {0, angleFrontRight} + Pi / 2], rotBody, posWheelFrontRight + posBody],
Text["Angle = " <> ToString[-angleFrontRight 180 / Pi] <> "°" <>
  "\nSpeed = " <> ToString[wheelFrontRightRadius / bodyRadius],
 RotationTransform[rotBody, posWheelFrontRight + posBody][
  posWheelFrontRight + 3 {1, 0} + posBody]],

Rotate[Circle[posWheelBackRight + posBody, circleRadius,
  {0, angleBackRight} + Pi / 2], rotBody, posWheelBackRight + posBody],
Text["Angle = " <> ToString[-angleBackRight 180 / Pi] <> "°" <>
  "\nSpeed = " <> ToString[wheelBackRightRadius / bodyRadius],
 RotationTransform[rotBody, posWheelBackRight + posBody][
  posWheelBackRight + 3 {1, 0} + posBody]],

Rotate[Circle[posWheelFrontLeft + posBody, circleRadius,
```

```
    {0, angleFrontLeft} + Pi / 2], rotBody, posWheelFrontLeft + posBody],
  Text["Angle = " <> ToString[-angleFrontLeft 180 / Pi] <> "°" <>
    "\nSpeed = " <> ToString[wheelFrontLeftRadius / bodyRadius],
   RotationTransform[rotBody, posWheelFrontLeft + posBody][
    posWheelFrontLeft + 3 {-1, 0} + posBody]],

  Rotate[Circle[posWheelBackLeft + posBody, circleRadius, {0, angleBackLeft} + Pi / 2],
   rotBody, posWheelBackLeft + posBody],
  Text["Angle = " <> ToString[-angleBackLeft 180 / Pi] <> "°" <>
    "\nSpeed = " <> ToString[wheelBackLeftRadius / bodyRadius],
   RotationTransform[rotBody, posWheelBackLeft + posBody][
    posWheelBackLeft + 3 {-1, 0} + posBody]]

}
```

Master 4WD/4WS vehicle control algorithm v1 - Graphics.nb

Initialize the positions and directions. Reset the controller

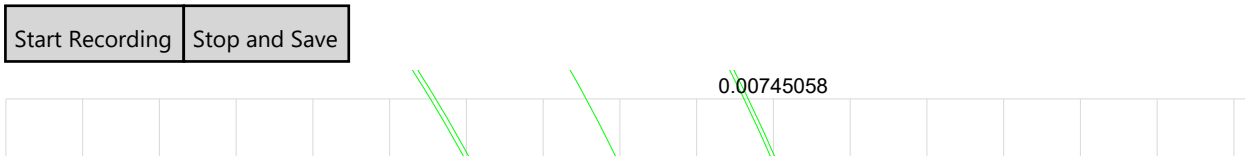
In[41]:=
```
(* In order to avoid dividing by zero,
a zeroRange was defined.  Any number that falls between -
 zeoRange and +zeroRange will be calculated separately.*)
zeroRange = 0.00001;

(*This is to reset the controller.  When this line is run,
the rest of program acts as if the controller was pulled out then plugged back in.*)
{x1, y1, z1, x2, y2, z2} =
  latest = ControllerState["SpaceNavigator", {"X1", "Y1", "Z1", "X2", "Y2", "Z2"}];
(*If z2 is really zero, then the car does not show the arrows because
 the routine assume it is on a curvature of some sort.  So,
create a very slight rotation so that the arrows show up.  Without this line,
there are not arrows until the controller experiences some rotation.*)
latest[[-1]] += zeroRange / 100;

(*create a routine to store last ten positions*)
posBodyHistory = Table[{x1, x2}, 10];
(*default initial direction of movement of the body*)
dirBody = {0, 1.};
(*default last direction of movement of the body.  Last direction
 is important because if the car didn't move between two Dynamic,
then the newly calculated direction would result in no direction.  In that case,
the old direction has to be used to poin the car to the direction it was pointing to.*)
dirBodyLast = {0, 1.};
(*The direction of the body's movement and how the car is rotated are two
 seperate matters in 4WD/4WS. rotBody is where the car's front is pointing to,
not where it is heading to.  Zero is north.*)
rotBody = 0.;
(*The controller's reading is so low that they have to be
 boosted.  Since translation and rotation produces different scale of numbers,
or they are used different on the program, they require different boost
 factors.  Notice that 0.5 was to used to make the rotation not so sensitive.*)
factor = 100000 {2, 2, 2, 1, 1, .5};
(*Where the car was located initially*)
posBody = {x1, y1};
```

This is the main routine that loop through

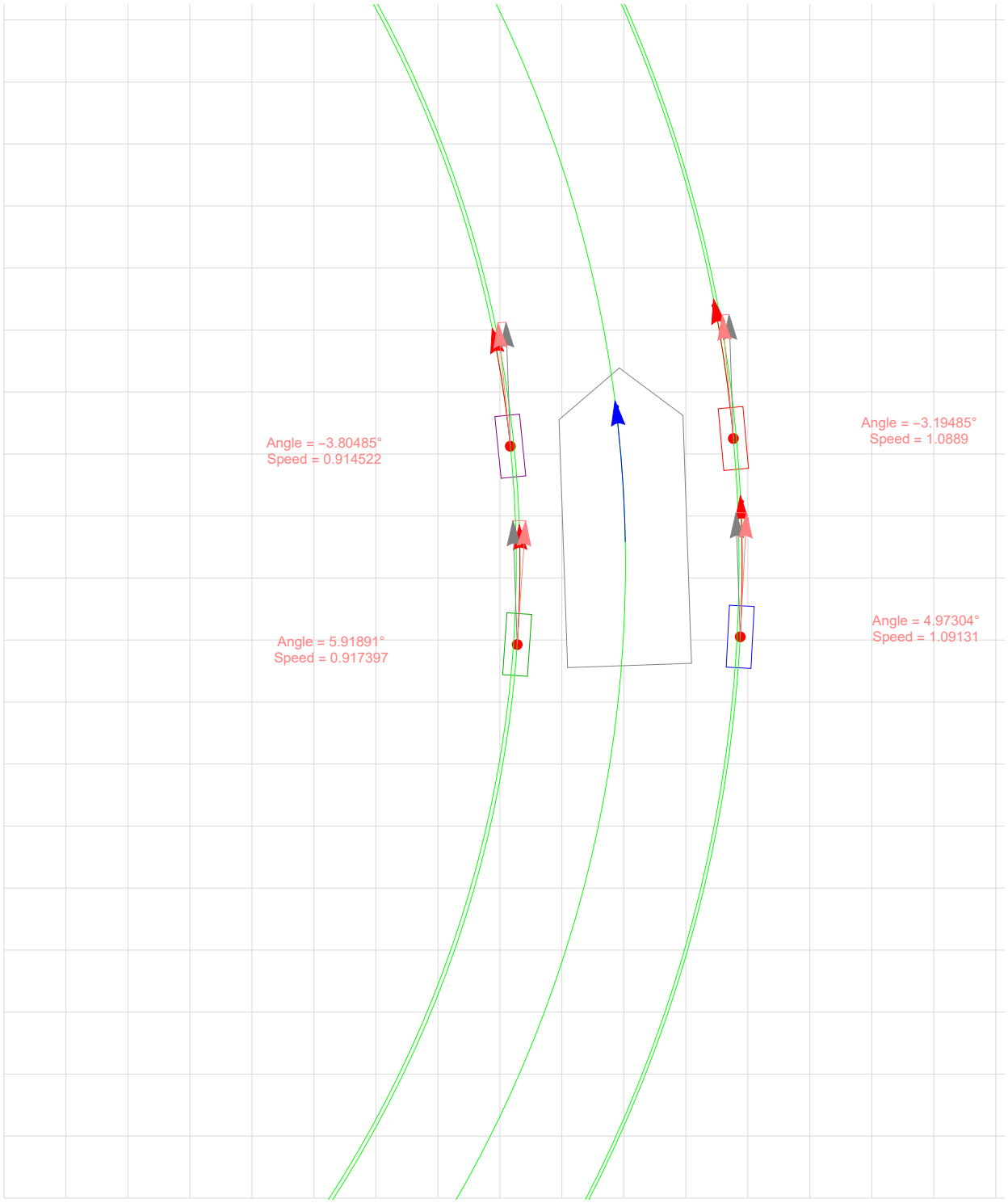
In[50]:=
```
Dynamic[
 (* read in the fresh input from the space mouse,
```

```
subtract the "latest" to get only the values that changed since the "latest".  Then
 multiply by the factor to adjust the sensitivities of each degree of freedom.*)
{x1, y1, z1, x2, y2, z2} = factor
   (ControllerState["SpaceNavigator", {"X1", "Y1", "Z1", "X2", "Y2", "Z2"}] - latest);
(* read the buttons to be used in case there is a rotation*)
{b1, b2} = ControllerState["SpaceNavigator", {"B1", "B2"}];

(* This is to collect last ten inputs.  posBodyHistory stores
 the lat readings.  RotateRight moves all the numbers to the right,
shifting them to the past by one step.*)
posBodyHistory = RotateRight[posBodyHistory];
(* then it fills the most recent spot with the latest data*)
posBodyHistory[[1]] = {x1, y1};

(*Now the vehicle has to travel along the circle.*)
(* Check how long it moved in x and y direction.  The x1 and
 y1 is the reading of the spacemouse.  The mouse's direction agrees
 with the vehicle only when the car is facing north.  Other times,
the intended direction such a forward, given by the driver has to be
 translated as the direction on the simulation by rotating it.  This step
 is necessary only in the screen.  The actual vehicle should not do this.*)
{dx, dy} = RotationTransform[rotBody][posBodyHistory[[1]] - posBodyHistory[[2]]];
(* Find out how far it traveled by finding the hypothenuse *)
distance = Norm[{dx, dy}];
(* Find out the radius of curvature of the motion so
 that the car's trajectory along that circle can be calculated. *)
bodyRadius = Norm[posCurveCenter];
(* now that we know how much it moved since last time "distance" and the radius of
 the circle "bodyRadius" we can calculate how many radians that motion is on the
 circle.  This is necessary because we cannot let the vehicle move straight.  We need
 to make it follow the circle because it has to follow the radius of curvature *)
dTheta = Sign[z2] distance / bodyRadius;
(* find out at what angle the center of the vehicle is
 relative to the center of curvature.  Or find out at what angle
 the center of curvature is relative to the vehicle's center. *)
theta = ArcTan @@ posCurveCenter;
(* now the car moves along the circle by moving tangential to the
 circle of curvature.  {-Sin[theta],Cos[theta]} is the perpendicular
 direction to a circle that is drawn with {Cos[theta],Sin[theta]} *)
posBody += -Sign[z2] distance {-Sin[theta], Cos[theta]};
(* Since the vehicle is going around a circular path,
the vehicle would appear to be spinning when seen from the top *)
rotBody += dTheta;
(* if b1 is pressed, then spin left. *)
If[b1, rotBody += 0.05;];
(* if b1 is pressed, then spion right *)
If[b2, rotBody -= 0.05;];

(* if the spacemouse was not touched,
which is detected by the size of the disatnce, then keep the last position *)
If[distance < 0.01, dirBody = dirBodyLast,
```

```
  (* but if it was touched then the direction becomes
   {dx, dy} and the last direction is also stored as {dx, dy}*)
  dirBody = dirBodyLast = {dx, dy}];
 (* adjust which direction  *)
 rotBodyDirectionAngle = ArcTan @@ dirBody + Pi / 2 + Pi;
 (* this is where the center of circular path
  is.  This is computed only when there is any z2 rotation. *)
 posCurveCenter = If[-zeroRange < z2 < zeroRange,
    (* if z2 was too small, then just assume that
     the center of the circular path is 10000 units away to the right,
    the rotate it for the case when it is not directly on the side of the vehicle. *)
    RotationTransform[rotBodyDirectionAngle, {0, 0}][{10000, 0}],
    (* if z2 value was not near zero, then calculate the center of circular path *)
    RotationTransform[rotBodyDirectionAngle, {0, 0}][{-1 / z2, 0}]];

 (*now comes actual painting of the simulation*)
 Column[{Row[{Button["Start Recording", startTime = AbsoluteTime[];
        recordMoves = True;
        fname = FileNameJoin[{NotebookDirectory[], "testfile.csv"}];
        stream = OpenWrite[fname];
        WriteString[stream, "Time, Center Speed, Front Right Speed, Back Right
           Speed, Front Left Speed, Back Left Speed, Front Right Angle,
    Back Right Angle, Front Left Angle, Back Left Angle\n"];],
       Button["Stop and Save", recordMoves = False;
        Close[stream]]}], Graphics[{(*Draw the body.comCar needs to know where
        the center of the body is,and what the rotation is*)comCar[posBody, rotBody],
       (*Draw the wheels.It needs to know where the body is,how rotated the body is,
       and where the center of circular path is because the wheels will align in the
        tanglential direction to that circular path*)comWheels[posBody, rotBody,
        If[b1, -0.001 posCurveCenter, If[b2, 0.001 posCurveCenter, posCurveCenter]]],
       (*The command below is to draw the lables for each wheel.*)(*comWheelLabels[
        {x1,y1},rotBody],*)(*Draw the trajectory which are all circles.Even straight
        lines are circles with very large radii.*)comDrawCircles[posBody, rotBody,
        (*if button 1 is pressed,then assume that the center of the circle is almost at
         the center of the body of the vehicle but not quite.It is at 0.001 radius from
         the center to preserve the direction of rotation*)If[b1, -0.001 posCurveCenter,
          (*On the other hand,if button 2 is pressed then it should spin right*)If[b2,
           0.001 posCurveCenter, posCurveCenter]]], (*Finally compute all 8 parameters
        (angles and speed for each wheel) and then draw the values on the graphcis.*)
       comDrawCalculations[posBody, rotBody, If[b1, -0.001 posCurveCenter, If[b2,
          0.001 posCurveCenter, posCurveCenter]]], comBackground, PlotLabel → distance]}]
If[distance > 0.001 && recordMoves, WriteString[stream,
    StringTake[ToString[{AbsoluteTime[] - startTime, distance, speedFrontRight,
        speedBackRight, speedFrontLeft, speedBackLeft, angleFrontRight,
        angleBackRight, angleFrontLeft, angleBackLeft}], {2, -2}] <> "\n"]]
]
```



Out[50]=




```
  WriteString[stream, StringTake[ToString[{AbsoluteTime[] - startTime, distance,
        speedFrontRight, speedBackRight, speedFrontLeft, speedBackLeft,
        angleFrontRight, angleBackRight, angleFrontLeft, angleBackLeft}], {2, -2}] <>
]]
```

In[51]:=
```
Dynamic[{z2, posCurveCenter}]
```

Out[51]= {0.0485278, {−20.6036, −0.363166}}

In[52]:= **Dynamic@distance**

Out[52]= 0.00745058