Appendix G: Miscellaneous Problems Encountered

I. PROBLEMS ENCOUNTERED WHILE VERIFYING PLAYER AND GAZEBO

I.A. Gazebo "`Namespace prefix ... is not defined`" errors

While attempting to verify Player and Gazebo using the packaged "simplecar" model, the author encountered a number of "`Namespace prefix ... is not defined`" errors.

Through review of the Gazebo mailing list archives, the author determined these errors were caused by missing `<xmlns>` declarations at the beginning of the file defining the packaged simplecar model, file `simplecar.model`. Including these declarations resolved the problem.

The author was unable to determine why some users of Gazebo were able to load this model without modification, as reported on the Gazebo mailing list, or why the packaged simplecar model was not revised to correct the problem when it was first reported on September 9, 2009. The author submitted patch number 2934729 to resolve the "`Namespace prefix ... is not defined`" errors on January 29, 2010.

I.B. Player "`Unhandled message for driver device`" error

While attempting to verify Player and Gazebo using the packaged examples, the author encountered an "`Unhandled message for driver device`" error. While this error occurred, the simplecar model would move slightly, then suddenly come to a stop, with unpredictable results. Sometimes the rear end of the vehicle would bounce into the air, and sometimes the vehicle would simply stop. After coming to rest, the model would no longer respond to commands. The author eventually identified the cause of this problem. See paragraph I.E.

While searching the Gazebo source code to determine the source of the error, the author noted the svn version of Gazebo revision 8443 included two copies of an 11.8 MB file in directories "`examples/player/ptz/.player`" and "`examples/player/ptz/.svn/text-base/.player.svn-base`" in which this error is repeated thousands of times. The compressed size of the Gazebo source distribution ("`gazebo-0.9.0.tar.bz2`") originally downloaded by the author, and which would not compile, was 17.4 MB. The copy of this file in directory "`examples/player/ptz/.svn/text-base/.player.svn-base`" was not included in the source distribution, but the copy in directory "`examples/player/ptz/.player`" was included in the source distribution.

I.C. ODE "`bNormalizationResult`" error

While attempting to verify Player and Gazebo using the packaged simplecar model, the author encountered the following ODE error:

```
ODE INTERNAL ERROR 1: assertion "bNormalizationResult"
failed in _dNormalize3()
[../../../include/ode/odemath.h] Aborted
```

Based on a review of ODE documentation ([44]), the author proposed this problem was related to the order of bodies in "hinge2" `<joint>` declarations in file `simplecar.model`, specifically a mismatch between bodies and the `<anchor>` declaration of the joint. ODE documentation states function `dJointGetHinge2Anchor`, corresponding to the file `simplecar.model` `<anchor>` declaration "returns the point on body 1", and that function

dJointGetHinge2Anchor2 “returns the point on body 2”. However, in the packaged version of file simplecar.model, the <anchor> declaration refers to the “left_front_wheel” and the <body1> declaration refers to the “chassis_body”. The author therefore reversed the order of the bodies in the <joint> declaration.

When the order of the bodies was reversed, the model would load successfully. The author submitted patch number 2934693 to reverse the order of the bodies in the <joint> declaration on January 19, 2010. However, the author was unable to interact with the model using the playerv utility at this time.

While attempting to resolve this problem, the author posted a message reporting the above to the Gazebo mailing list, and received the following reply: “I've seen bNormalizationResult in the past due to float/double inconsistency between ode and gazebo.” ([101]). As a result, the author uninstalled ODE in accordance with Appendix C and re-installed ODE in accordance with Appendix B using the “--enable-double-precision” and “--enable-demos” flags. The use of the “--enable-double-precision” flag to enable double precision in ODE resulted in errors when attempting to run Gazebo. The author then uninstalled ODE in accordance with Appendix C and re-installed ODE in accordance with Appendix B using the “--enable-demos” flag and confirmed that the ODE demos would run without errors.

Finally, the author uninstalled ODE in accordance with Appendix C and re-installed ODE in accordance with Appendix B using the “--enable-demos” and “--disable-asserts” flags to disable assertion checking. The author was unable to

determine why installation instructions for Gazebo do not require the use of the "`--disable-asserts`" flag when compiling ODE for use with Gazebo. As a result, the author's only conclusion is that a default installation of Gazebo using a default installation of ODE will fail due to ODE assertions when attempting to load the packaged simplecar model.

I.D. Playerv "Devices>position2d" menu

As noted in paragraph I.C., the author was unable to interact with the packaged simplecar model using the `playerv` utility. The simplecar model would move slightly, then suddenly come to a stop, with unpredictable results. Sometimes the rear end of the vehicle would bounce into the air, and sometimes the vehicle would simply stop. After coming to rest, the model would no longer respond to commands.

The author initially proposed the problem was due to the interaction between the disabling of a menu item via function `rtk_menuitem_isactivated` in file `rtk_menu.c` and function `position2d_create` in file `pv_dev_position2d.c`. File `pv_dev_position2d.c` creates menu items "Enable" and "Disable" on the "Devices>position2d" menu in the `playerv` utility which appeared to be disabled by default (i.e., no checkbox is available to enable or disable the position2d interface), and file `rtk_menu.c` appeared to disable menu items as soon as they were enabled via the following lines:

```
if (item->activated)
{
  item->activated = FALSE;
```

```
  return TRUE;
}
```

The author revised file `pv_dev_position2d.c` to enable the menu items to be enabled (i.e., to display a checkbox indicating their status). The author was then able to verify the device was enabled as expected and remained enabled while attempting to command the simplecar model using the `playerv` utility.

Following resolution of the Gazebo `ODEHingeJoint.cc` error (see paragraph I.E.), the author discovered the menu items "Enable" and "Disable" on the "Devices>position2d" menu in the `playerv` utility can be enabled despite not having an associated checkbox and that the `playerv` utility cannot be used to command a model using the position2d interface unless the space next to the "Enable" interface option in the "Devices>position2d" menu, where a checkbox would be if the "Enable" interface option were enabled, is first clicked on. The author was unable to determine why the "Enable" and "Disable" options appear to be disabled.

I.E. Gazebo "`ODEHingeJoint.cc`" error

As noted in paragraph I.C., the author was unable to interact with the simplecar model using the `playerv` utility. Initially, the author proposed the cause of the problem was the disabling of the position2d interface as soon as it was enabled. The author determined that although a problem exists (to enable the position2d interface, users must click a non-existent checkbox next to "Enable" in the "Devices>position2d" menu), the problem did not cause the observed behavior. See paragraph I.D.

The author has been monitoring the "playerstage-developers", "playerstage-

gazebo", and "playerstage-users" mailing list since beginning this research. Another user identified one of the two causes of this problem the author has been able to verify, and submitted bug report number 2933700 on January 17, 2010 to report it to the Player Project. The order of parameters passed to function "SetParam" in file `ODEHingeJoint.cc`, function:

```
void ODEHingeJoint::SetVelocity(int /*index*/, double
angle)
{
  this->SetParam(angle, dParamVel);
}
```

was reversed. The function should be:

```
void ODEHingeJoint::SetVelocity(int /*index*/, double
angle)
{
  this->SetParam(dParamVel, angle);
}
```

The author reversed the order of parameters as described and was able to verify Player and Gazebo using the packaged examples, specifically the simplecar model.

II. PROBLEMS ENCOUNTERED WHILE VALIDATING THE IMPROVED CONTROLLER

II.A. Gazebo `ODEHinge2Joint::GetAngle` and `GetVelocity` problem

While validating the improved controller, the author encountered a problem when setting the angular velocity of the steering wheels around the steering axis. Under certain

circumstances, the angular velocity of a steering wheel, typically the outside wheel, would increase suddenly due to a large difference between the target steering angle and current steering angle. This caused the steering wheel to suddenly turn at high speed around the steering axis. The effect of friction would then cause the wheel to appear to "dig in" and throw the rear of the model into the air.

Although this behavior is consistent with expectations from the perspective of physical realism, the purpose of the improved controller was to effectively limit the maximum angular velocity at model CG to prevent the model from being able to turn at a rate faster than allowed by representative challenge vehicle and course geometry and thereby prevent rollover. The purpose of the improved controller was to prevent exactly this problem.

The angular velocity of each steering wheel around the steering axis is determined by the difference between the target steering angle and current steering angle of the wheel and the update rate of the improved controller. Multiplying the difference by the update rate yields the angular velocity which must be applied to reach the target steering angle in one controller time step. The controller limits the angular velocity to the calculated angular velocity or maximum angular velocity at model CG, whichever is less.

The author originally proposed this problem was caused by Gazebo function `ODEHinge2Joint::GetAngle`, which is a wrapper around ODE function `dJointGetHinge2Angle1`, which itself is a wrapper around ODE function `dxJointHinge2::measureAngle`. This function returns the value of the C programming language function `atan2`, which is undefined if both arguments are equal

to zero. The author noted that the problem occurred at approximately the same time and position in each trial, and proposed it was related to the difference between the target steering angle and current steering angle of the steering wheel approaching zero because the value returned by function `ODEHinge2Joint::GetAngle` increased suddenly as the current steering angle approached the target steering angle in each trial.

The author attempted several solutions to this problem: revising the time step, reducing the mass of the representative challenge vehicle, and setting the velocity of the chassis body directly during each time step. Each proposed solution was rejected as unrealistic or problematic.

The author then revised improved controller function `Wheel::Update` to store the last known value returned by function `ODEHinge2Joint::GetAngle` in a private class member variable and use it in lieu of the value returned by the function itself when the current steering angle approached the target steering angle, but this did not resolve the problem. As a result, the author concluded function `ODEHinge2Joint::GetAngle` was not the cause of the problem and that the sudden increase in angular velocity observed was an indirect effect of another problem.

The author determined the problem was caused by a combination of inaccurate maximum angular velocity determination and insufficient torque. Diagnosing the problem was made more difficult by lack of a return value from Gazebo function `ODEHinge2Joint::GetVelocity`.

The packaged controller subtracted the angular velocity returned by function `ODEHinge2Joint::GetVelocity` from the target velocity determined by the

controller every time the controller was updated. However, it is unclear if this was ever successful because this function had no return value.

After revising function `ODEHinge2Joint::GetVelocity` to return the angular velocity of the joint the author was able to use this function to determine that the wheel joints were unable to reach the target angular velocities set by the improved controller before the next controller update. As a result, the difference between the target angle and current angle (returned by function `ODEHinge2Joint::GetAngle`) increased as the simulation ran. The controller was setting the target angular velocity based on the difference between the target steering angle and current steering angle, which was large. Multiplying this difference by the update rate of the controller further increased the resulting value. The controller was therefore attempting to set a target velocity that far exceeded the current angular velocity, but was unable to achieve the target steering angle in one controller time step.

The ODE Manual states: “The preferred method of setting body velocities during the simulation is to use joint motors. They can set body velocities to a desired value in one time step, provided that the force/torque limit is high enough.” ([44]). When the author increased the torque applied to the model's joints to 10,000, the joints were able to reach their target steering angle in one controller time step, eliminating the large difference between their current steering angle and target steering angle. However, although this resolved the problem by making it possible for the joints to reach their target steering angle in one controller time step, this resolution did not address the root cause of the problem, which was inaccurate maximum angular velocity determination.

When determining the maximum angular velocity at model CG in a turn of arbitrary radius, the author failed to include a factor of 2π in the denominator. As a result, the maximum angular velocity was approximately six times larger than it should have been. This problem did not become apparent until the second test, because the improved controller correctly limited the maximum steering angle of the model during the first test to that allowed by the vehicle's turning circle, and because the author allowed the steering wheels to turn to their maximum right extent before accelerating the model. Because the controller limits the angular velocity of each steering wheel around the steering axis to the calculated angular velocity or maximum angular velocity at model CG, whichever is greater, this had the effect of allowing the controller to set an angular velocity which exceeded the maximum angular velocity which would have prevented this problem.

In combination with increasing the torque applied to the steering axis so that the steering wheels were able to reach their target steering angles in one improved controller time step, correctly calculating the maximum angular velocity effectively resolved this problem. The author eliminated the use of function `ODEHinge2Joint::GetVelocity` entirely after noting that it occasionally returns "`nan`" (literally "not a number"), and proposes this may be the reason the function had no return value originally.

II.B. Swaybar implementation

While validating the improved controller, the author encountered a problem with rollover at moderate speeds, when the controller was correctly setting the velocity of each wheel so that rollover should have been prevented. Based on a review of the ODE

Manual, the author noted that similar problems were reported by other users, and that a proposed solution was to implement “anti-sway” bars to limit the back-and-forth rotation of a model around the x-axis.

The author revised the improved controller to calculate and apply a force to each joint during each controller update to compensate for some of this motion. However, the calculated displacement for each joint during each controller update was near zero, and the force applied to each joint insignificant. The author concluded the swaybar implementation was not a solution to the problem observed.

Through review of the model, the author determined that two of the representative challenge vehicle characteristics in use by the controller were in error.

Prior to deciding to develop an improved controller to achieve stability at high speeds, the author used the vehicle characteristics for a 2009 Honda Accord as the basis for a model using the packaged controller. The mesh for this model was also used as the “Car Obstacle”. See Figure 8.

The author was originally unable to determine the SSF of the Team 2005-06 challenge vehicle while researching the vehicle through commercial used car search services ([102] and [103]). As a result, the author estimated the Team 2005-06 challenge vehicle SSF using available information about the vehicle and general information about the class of vehicle.

When revising the 2009 Honda Accord model to simulate the representative challenge vehicle, the author calculated the height of vehicle CG using the alternate SSF and placed a geom having a mass of 1720 kg at a height of 0.894 m as the chassis body of

the model. This was an error. The author later determined the Team 2005-06 challenge vehicle had an SSF of 1.17 ([104] and [105]).

To eliminate other potential errors, the author reviewed all representative challenge vehicle characteristics in use by the improved controller, and noted one additional error: when the author converted from English to Metric units, the author incorrectly calculated the rear track width of the model as 1.580 m (62.2 in), in lieu of 1.529 m (60.2 in), due to an error when entering data.

As a result of these errors, the effective SSF of the model was 0.88, but the improved controller was calculating the maximum velocity of the model using a SSF of 1.17, and accelerating the model to a velocity which predictably resulted in rollover. The author re-calculated the height of model CG using a SSF of 1.17 (0.653 m), and revised the model XML file to correct these errors. The model was then able to successfully complete the turn.

The author considers this incident, more than any other, highlights the ability of ODE as an accurate physics simulation to help identify problems with world files, models, or controller logic which are intended to model real-world interaction, and to help model real-world interaction which cannot be realistically evaluated, e.g., the risk of rollover.

III. PROBLEMS ENCOUNTERED DURING EVALUATION OF THE SIMULATION TARGETS

III.A. "ODE Message 3" error

The author encountered the following ODE error when running the simulations:

`ODE Message 3: LCP internal error, s <= 0 (s=...)`

with different values for `s`. The ODE Manual states this error "is usually caused by an object ramming into another with too much force (or just the right force)." ([44]), and recommends decreasing the mass of the object or changing the simulation timestep to eliminate the error, but also states: "this [error] won't crash your simulation".

The mass of the model was based on the mass of the representative challenge vehicle and the simulation timestep was selected through a process of trial and error to produce a stable simulation. As a result, the author decided not to change the mass of the model or the simulation timestep, and chose to ignore the error as a warning.

III.B. Angular unit inconsistencies between Player and Gazebo

The Player Project "World File Syntax" states: "Unless otherwise specified, the world file uses SI units (meters, seconds, kilograms, etc). One exception is angles and angular velocities, which are measured in degrees and degrees/sec, respectively." ([106]).

However, the `playerv` utility returns angular values in radians and angular velocities in radians/s. In addition, ODE functions in use by the improved controller such as `SetVelocity` for ODEHingeJoint and ODEHinge2Joint joints expect radians/s.

The ODE Manual states: "...ODE doesn't use specific units. You can use anything you wish, as long as you stay consistent with yourself." ([44]). Although ODE is unit agnostic, SI units are recommended.

For consistency with Gazebo world files, the improved controller reads Euler angles (included in `<rpy>` declarations) and the maximum steering angle (included in `<steerMaxAngle>` declarations) from an XML file which are measured in degrees,

but internally uses angles and angular velocities measured in radians and radians/s.

III.C. `ODEHinge2Joint <anchorOffset>` problem

The author encountered a number of problems when evaluating the packaged steering controller. Specifically, the author was unable to interact with the simulation until the `AutoDisableFlag` and `SetParameters` problems were resolved. However, while attempting to resolve these problems, the author reviewed the ODE Manual and determined that the order of the chassis and wheel bodies defined by the `<joint>` declarations in each joint's `<body1>` and `<body2>` declarations was reversed in some joints but not others.

Specifically, “body1” of joints “left_front_wheel_hinge” and “right_front_wheel_hinge” was “chassis_body”, but “body1” of joints “left_rear_wheel_hinge” and “right_rear_wheel_hinge” was the corresponding wheel. In the example included with the ODE Manual, “body1” is the chassis and “body2” is the wheel. The author attempted to resolve the problem by revising the order of the body declarations so that “body1” was “chassis_body” and “body2” the corresponding wheel for all joints. This was unsuccessful.

When the order was reversed so that “body1” was the chassis of the vehicle in lieu of the wheel for all joints, the author observed a “wobble” when the “simplecar” model was driven around in simulation. The wobble had the effect of causing the wheels of the model to leave the ground at high speed. Because the wobble of the wheels was not synchronized, the model was very unstable and would readily roll over. Without a stable model capable of traveling at speeds typical of vehicles participating in the 2004

and 2005 GCE in simulation, the author would not be able to test the rollover condition or effectively evaluate LIDAR.

Initially, the author believed the wobble was due to a mismatch between the `<anchorOffset>` declaration and the body to which it referred, and attempted to resolve this problem by revising the `<anchorOffset>` declaration for each wheel to be relative to the chassis of the vehicle, not the wheel. This was unsuccessful.

At this point, the author requested clarification from the playerstage-gazebo mailing list and received conflicting reports that the "simplecar" model and steering controller were and were not working. Specifically, that another user was able to load the model, but had not confirmed the model could be controlled using the `playerv` utility. Based on a response, the author downloaded Robot Operating System (ROS) ([107]) using the `svn` utility for the purposes of evaluating it for use.

The author was unable to locate the equivalent ODE source files in the ROS package "core code", and did not want to download and install software which might interfere with research to date. Rather than return to the beginning, install ROS, and resolve problems similar to those encountered when installing Player and Gazebo, the author elected not to continue with ROS.

While attempting to determine the cause of the wobble, the author reviewed the ODE Manual and noted that similar problems due to off-axis rotation at high-speed have been reported. ODE provides functions to limit the off-axis rotation of a body: `dBodySetFiniteRotationMode` and `dBodySetFiniteRotationAxis`. The author implemented these functions in files `Body.cc` and `ODEBody.cc` (and

corresponding header files) to utilize the ODE-provided functions in an attempt to limit off-axis rotation: `SetFiniteRotationMode` and `SetFiniteRotationAxis`.

As a result of these changes, the author was able to read the values of two parameters from the model XML file: `finiteRotationMode` and `finiteRotationAxis` and set the finite rotation mode and axis of a body. However, setting the finite rotation mode and axis of the wheel bodies did not eliminate the wobble.

At this point, the author hypothesized that the problem was caused by miniscule errors in calculation over thousands of simulation cycles of the physics engine due to the weight of the chassis (1720 kg), as ODE attempted to maintain the position of the chassis body relative to each wheel. To eliminate the possibility that the length of the joints was contributing to the problem observed, the author revised the model to eliminate the z-dimension of the anchor offset by loading the model with the wheels at the same height as the chassis body. As a result of this change, the author was able to identify the cause of the wobble by experimentation.

The body of each wheel rotated in a plane at a fixed distance from its defined axis (y-axis for the rear wheels and z-axis for the front wheels). By eliminating the z-dimension anchor offset, the author changed the distance from the axis around which the body rotated, resulting in clear rotation around the y-axis. By increasing the distance from the axis, the effect was more pronounced.

As a result, the author reviewed the ODE Manual to determine what the intended effect of the anchor offset was, and discovered that ODE does not provide a function to set or return an anchor offset parameter. The anchor offset is used by file `Joint.cc`

(and corresponding header file) for “setting anchor relative to gazebo body frame origin”. However, the anchor offset is applied to the body when the model is loaded. As a result, subsequent attempts to rotate the body around an axis rotate the offset body.

ODE attempts to keep the bodies of a joint together. Small values for anchor offset had less effect on the “simplecar” model than larger values, and extremely small values had no observable effect. When the values for anchor offset were too large, and one axis of rotation was eliminated, it became clear ODE was no longer able to compensate for the forces being placed on the axis by the controller, and was allowing the wheel bodies to freely rotate at a fixed distance from their axes, as defined by the `<anchorOffset>` declaration for each joint in the “simplecar” model.

In addition, the author determined the effect of changes to the x-, y-, and z-dimensions of the anchor offset in each joint by experimentation. The author concluded the orientation of the anchor offset was not preserved when the `<anchorOffset>` declaration was used. The wheels of the model were created from a cylinder, a built-in type of geom. Gazebo's built-in cylinder geom is defined by a radius and height. The height of the cylinder extends along the positive z-axis. To create a wheel, the cylinders modeling the left wheels were rotated around the x-axis by 90 degrees by the wheel body's `<rpy>` declaration when the model was loaded. Based on the behavior observed, if an anchor offset is declared for the wheel's corresponding joint this rotation also rotates the axis around which the wheels rotate by 90 degrees so that the anchor offset's positive y-dimension becomes the positive z-dimension, and positive z-dimension becomes the negative y-dimension. A similar rotation was observed for the

cylinders modeling the right wheels. As a result, a front wheel would revolve around the z-axis by the anchor offset's positive y-dimension, and around the y-axis by the anchor offset's negative z-dimension. This made the problem more difficult to resolve.

The author considers this may be the cause of the failure of the front wheels of the "simplecar" model to turn when a type of "full" was declared and the packaged steering controller was in use. The revolution of the wheel bodies around the y- and z-axes may effectively "bind" the wheels, preventing rotation. The author eliminated the `<anchorOffset>` declaration from each joint, and the use of anchor offset entirely. This greatly increased stability at high speed by eliminating the wobble, and made it possible for the author to implement four-wheel drive at speeds typical of vehicles participating in the 2004 or 2005 GCE.

The author considers this and other similar issues, such as the reversed order of parameters described above, to be evidence of a "meandering direction of development" evident through review of the Gazebo codebase. The Gazebo codebase is being actively developed. Some features have been abandoned, others were never fully implemented, and the purpose of some functions is unclear from function declarations.

For example, ODE has no intrinsic function to set a second hinge anchor in either a `Hinge` or a `Hinge2` joint. However, Gazebo functions `ODEHingeJoint::SetAnchor` and `ODEHinge2Joint::SetAnchor` both include a parameter `index`, which is commented out. Parameter `index` is commented out in several other `ODEHingeJoint` functions.

In addition, the ODE Manual states: "These parameter names can be optionally

followed by a digit (2 or 3) to indicate the second or third set of parameters, e.g. for the second axis in a hinge-2 joint, or the third axis in an AMotor joint." ([44]). However, the list of parameters to which the ODE Manual refers does not include parameter `axis`. Functions for setting or getting the axis of a `Hinge2` joint are documented by the ODE Manual, but not parameter `axis`. As a result, the author concluded ODE may have, at one time, used parameter names followed by a digit to indicate a second or third set of parameters, and that Gazebo functions `ODEHingeJoint::SetAnchor` and `ODEHinge2Joint::SetAnchor` may be referring to these numbers as the "index", and that "index" has since been commented out because Gazebo has not been updated to remove these references.

Diagnosing the `ODEHinge2Joint anchorOffset` problem required several days, during which more productive research was delayed.

III.D. `OGRE::AxisAlignedBox` error

The author encountered the following error while attempting to resolve the `ODEHinge2Joint <anchorOffset>` problem reported above (only a portion of the actual error message is included herein):

```
Assertion `(min.x <= max.x && min.y <= max.y && min.z
<= max.z) && "The minimum corner of the box must be
less than or equal to maximum corner"' failed.
```

In general, this error occurred immediately before a segmentation fault which terminated the running simulation. The author did not encounter this error after resolving the `ODEHinge2Joint <anchorOffset>` problem.