International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:11, No:6, 2017

Virtual 3D Environments for Image-Based Navigation Algorithms

V. B. Bastos, M. P. Lima, P. R. G. Kurka

Abstract**—**This paper applies to the creation of virtual 3D environments for the study and development of mobile robot image based navigation algorithms and techniques, which need to operate robustly and efficiently. The test of these algorithms can be performed in a physical way, from conducting experiments on a prototype, or by numerical simulations. Current simulation platforms for robotic applications do not have flexible and updated models for image rendering, being unable to reproduce complex light effects and materials. Thus, it is necessary to create a test platform that integrates sophisticated simulated applications of real environments for navigation, with data and image processing. This work proposes the development of a high-level platform for building 3D model's environments and the test of image-based navigation algorithms for mobile robots. Techniques were used for applying texture and lighting effects in order to accurately represent the generation of rendered images regarding the real world version. The application will integrate image processing scripts, trajectory control, dynamic modeling and simulation techniques for physics representation and picture rendering with the open source 3D creation suite - Blender.

Keywords**—**Simulation, visual navigation, mobile robot, data visualization.

I. Introduction

SIMULATORS are computerized environments which emulate some activities of real phenomena that users can manipulate, explore and experiment [1]. Nowadays to reduce the cost and time of research, scientists develop and use their own tools to create a simple and easy way of testing ideas, theories and scripts, without the need to depend physically on certain machinery and equipment.

Using simulations over real applications brings advantages to a project, of which can be cited:

- Repeatability – possibility of easier reproduction of a test innumerous times, saving time and other resources;
- Flexibility – less effort in the creation and modification of the desired work place;
- Enable the tests in unreachable or complex environments;
- Simulate dangerous situations that could possibly damage the prototype;
- Get exact values of position, orientation, velocity and acceleration without any extra device.

A number of different software packages for 3D simulation in robotics, like Gazebo (Fig. 1) [2], Weebots [3] and V-Rep [4]. These tools are made, basically, to reproduce the estimated dynamic model of a robot, interacting with their neighborhood using sensors and actuators.

Many experiments can be represented with these applications, but one problem is that their virtual camera sensor generally is very limited compared to the rendered images generated by computer graphics platforms, such as Blender [5]. Another issue is the reproduction of specific materials with reflection and deflection of light, if the software has an implementation; it is a very simplistic one. This can be a problem for visual odometry techniques, such as those developed in [6], [7]

The inability of these software to render precise patterns and complex light effects presented in the real world, reduce their reliability for the training of some algorithm, like neural-network based scripts.

The next section of this paper details the creation of a robot and environment, combined with the physics presented in an estimated dynamic model for differential robots. After the methodology is explained, some results obtained are discussed. Furthermore, some comments are included in the conclusion regarding the main future works applications of this research, and possible enhancements for this simulation.

II. Combination of Robot Dynamics with a Computer Graphics Platform

As explained in the previous section, the current simulators oriented to the field of robotics do not have the same flexibility to render images having complex light effects and materials with a difficult reproduction. However, they have a library with the estimated dynamic models implemented for many mobile robots, ready to use.

The main idea of this work is to combine the physics of a dynamic model with high quality rendering algorithms for camera simulation. This kinematics is the discussion for the first subtopic.

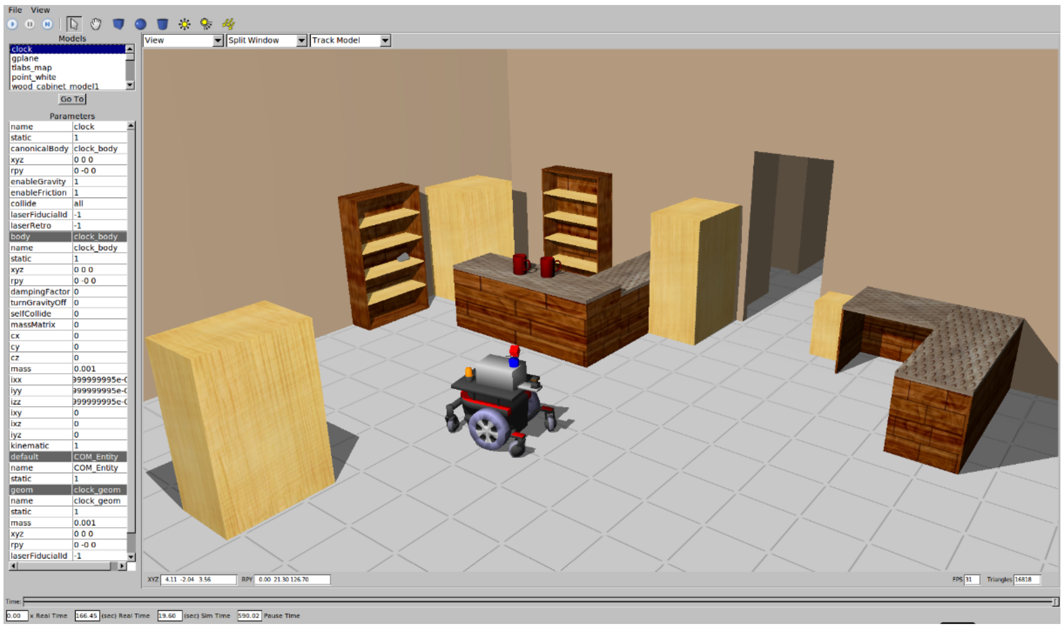
A. Dynamic Model for Differential Robots

Primarily, it is necessary to study the robot to implement its movements in the simulation. As the University of Campinas has the Pioneer 3DX available for experiments, the mobile robot chosen was the differential type. This model is the presented robot in Fig. 2 [6]. Fig. 3 [8] shows this robot view from the top, defining the positions of each variable for future uses in the paper. In this figure, u e ω are, respectively, the linear and angular velocities of the robot, G is its center of mass, C is the position of its castor wheel, h is the interest point (the coordinates x and y in the plain XY), ψ is the robot orientation, and a is the distance between the interest point and

V. B. Bastos and Lima, M. P. are PhD students in the Integrated Systems Department at University of Campinas, SP-Brazil (e-mail: bastos_est@hotmail.com, mvpleng@fem.unicamp.br).

Kurka, P. R. G is with the Integrated Systems Department at University of Campinas, SP-Brazil as a professor. (e-mail: kurka@fem.unicamp.br).

the center point of the virtual axis that connects the traction wheels (B point).



name	clock
static	1
canonicalBody	clock_body
xyz	0 0 0
rpy	0 -0 0
enableGravity	1
enableFriction	1
collide	all
laserFiducialId	-1
laserRetro	-1
body	clock_body
name	clock_body
static	1
xyz	0 0 0
rpy	0 -0 0
dampingFactor	0
turnGravityOff	0
selfCollide	0
massMatrix	0
cx	0
cy	0
cz	0
mass	0.001
ixx	999999995e-(
iyy	999999995e-(
izz	999999995e-(
ixy	0
ixz	0
iyz	0
kinematic	1
default	COM_Entity
name	COM_Entity
static	1
geom	clock_geom
name	clock_geom
static	1
mass	0.001
xyz	0 0 0
rpy	0 -0 0
laserFiducialId	-1

XYZ 4.11 -2.04 3.56 RPY 0.00 21.30 126.70 FPS 11 Triangles 16818

Time: 0.00 x Real Time 166.45 (sec) Real Time 19.60 (sec) Sim Time 590.02 Pause Time

Fig. 1 Example of a scene developed in Gazebo

Fig. 2 Pioneer 3DX

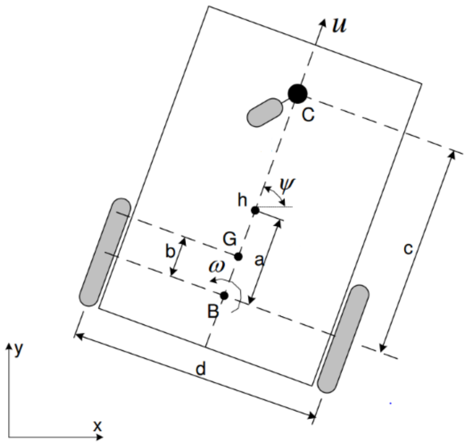


Fig. 3 Representation of differential robot with the desired variables

For dynamic models, the experiments use those developed by De La Cruz [8], based on the model proposed by Zhang et al. [9], which present as entry signals, the values of torque applied to the left and right wheels. However, commercial robots usually accept linear and velocity commands, and not torque inputs for its motors. In this context, in De La Cruz [8] proposed two models for mobile robots for the differential type. The first model has the entry signals as the tension applied in both motors of the left and right wheels, and in the last one, the references are the linear and angular velocities performed by the robot. The complete model for the first case is represented by (1):

$$\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\psi}\\ \dot{u}\\ \dot{\omega}\end{bmatrix}=\begin{bmatrix}\cos\psi & -a\sin\psi\\ \sin\psi & a\cos\psi\\ 0 & 1\\ \frac{\theta_3^0}{\theta_1^0}\frac{r^2\omega^2}{u} & -\frac{2u}{\omega}\frac{\theta_4^0}{\theta_1^0}\\ -2r^2\omega\frac{\theta_3^0}{\theta_2^0} & -\frac{\theta_4^0}{\theta_2^0}d^2\omega\end{bmatrix}\begin{bmatrix}u\\ \omega\end{bmatrix}+\begin{bmatrix}0 & 0\\ 0 & 0\\ 0 & 0\\ \frac{2r}{\theta_1^0} & 0\\ 0 & \frac{2rd}{\theta_2^0}\end{bmatrix}\begin{bmatrix}\frac{v_r+v_l}{2}\\ \frac{v_r-v_l}{2}\end{bmatrix}+\begin{bmatrix}\delta_x\\ \delta_y\\ 0\\ \bar{\delta}_\mu\\ \bar{\delta}_\omega\end{bmatrix} \quad (1)$$

where, $\delta=[\delta_x,\delta_y,0,\bar{\delta}_\mu,\bar{\delta}_\omega]$, is the uncertain vector associated with the mobile robot, in which δ_x and δ_y are functions of the sliding velocity and robot orientation, $\bar{\delta}_\mu$ e $\bar{\delta}_\omega$ are functions of the physical parameters like mass, inertia, wheel diameter, motor's parameter, wheel force, among others, and v_r and v_l, are the tensions applied in the left and right tires, respectively. The model parameters $(\theta_1^0,\theta_2^0,\theta_3^0,\theta_4^0)$ are represented by (2)-(5).

$$\theta_1^0=\frac{R_a}{k_a}(mr^2+2I_e)[V.s^2] \quad (2)$$

$$\theta_2^0 = \frac{R_a}{k_a}\left(I_e d^2 + 2r^2(I_z + mb^2)\right)[V.m^2.s^2] \tag{3}$$

$$\theta_3^0 = \frac{R_a}{k_a} mb[V.s^2/m] \tag{4}$$

$$\theta_4^0 = \frac{R_a}{k_a}\left(\frac{k_a k_b}{R_a} + B_e\right)[V.s/rad] \tag{5}$$

For the above equations, Ra $(ohms)$ is the motor electric resistance, kb $(V.s/rad)$ is its voltage constant, ka $(N.m/A)$ is its torque constant multiplied by the gear ratio, Be $(N.s/rad)$ is the rotational friction, m (Kg) is the robot total mass, Iz $(Kg.m^2)$ is the inertia at the G point, Ie $(Kg.m^2)$ is the inertia for each group of wheels, r (m) is the radius of the wheels, b (m) and d (m) are the distances represented in Fig. 3.

The first model is useful when it is possible to control directly the tensions in each robot's motor. However, commercial robots generally have internal controllers that receive velocities references for each one, and do not allow their tension to be directly controlled. In this context, in De La Cruz [8] considered that the internal controllers are PD (Proportional and Derivative), with proportional gains $k_{PT} > 0$ and $k_{PR} > 0$, and derivatives $k_{DT} \geq 0$ e $k_{DR} \geq 0$. Thus, it can be represented by (6):

$$\begin{bmatrix} v_u \\ v_\omega \end{bmatrix} = \begin{bmatrix} k_{PT}(u_{ref} - u) - k_{DT}\dot{u} \\ k_{PR}(\omega_{ref} - \omega) - k_{DR}\dot{\omega} \end{bmatrix} \tag{6}$$

In this case, v_u and v_ω are defined by (7) and (8):

$$v_u = \frac{v_r + v_l}{2} \tag{7}$$

$$v_\omega = \frac{v_r - v_l}{2} \tag{8}$$

The values of u and ω are defined by (9) and (10), respectively.

$$u = \frac{1}{2}[r(\omega_r + \omega_l)] \tag{9}$$

$$\omega = \frac{1}{d}[r(\omega_r - \omega_l)] \tag{10}$$

The angular velocities are represented by ω_r for the right wheel, and $\omega_l$for the left one. From (1), (6)-(10), we obtain (11):

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\psi} \\ \dot{u} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} \cos\psi & -a\sin\psi \\ \sin\psi & a\cos\psi \\ 0 & 1 \\ -\frac{\theta_4}{\theta_1} & \frac{\theta_3}{\theta_1}\omega \\ -\frac{\theta_5}{\theta_2}\omega & -\frac{\theta_6}{\theta_2} \end{bmatrix} \begin{bmatrix} u \\ \omega \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ \frac{1}{\theta_1} & 0 \\ 0 & \frac{1}{\theta_2} \end{bmatrix} \begin{bmatrix} u_{ref} \\ \omega_{ref} \end{bmatrix} + \begin{bmatrix} \delta_x \\ \delta_y \\ 0 \\ \delta_u \\ \delta_\omega \end{bmatrix} \tag{11}$$

where, u_{ref} and ω_{ref} are the reference signals for the linear and angular velocities, $\theta = [\theta_1\ \theta_2\ \theta_3\ \theta_4\ \theta_5\ \theta_6\ ]^T$ is the model parameters vector, and $\delta = [\delta_x\ \delta_y\ 0\ \delta_u\ \delta_\omega\ ]^T$ is the uncertain vector associated to the mobile robot model. As previously presented, δ_x and δ_y are functions of the sliding speed and orientation, however δ_u and δ_ω are functions of physical parameters.

The values of $\theta_1, \theta_2, \theta_3, \theta_4, \theta_5$ are represented by (12)-(17).

$$\theta_1 = \left[\frac{R_a}{k_a}(mr^2 + 2I_e) + 2rk_{DT}\right]\frac{1}{2rk_{PT}}[s] \tag{12}$$

$$\theta_2 = \frac{\left[\frac{R_a}{k_a}\left(I_e d^2 + 2r^2(I_z + mb^2)\right) + 2rdk_{DR}\right]}{2rdk_{PR}}[s] \tag{13}$$

$$\theta_3 = \frac{R_a}{k_a}\frac{mbr}{2k_{PT}}[s.m/rad^2] \tag{14}$$

$$\theta_4 = \frac{R_a}{k_a}\left(\frac{k_a k_b}{R_a} + B_e\right)\frac{1}{rk_{PT}} + 1 \tag{15}$$

$$\theta_5 = \frac{R_a}{k_a}\frac{mbr}{dk_{PR}}[s/m] \tag{16}$$

$$\theta_6 = \frac{R_a}{k_a}\left(\frac{k_a k_b}{R_a} + B_e\right)\frac{d}{2rk_{PR}} + 1 \tag{17}$$

The parameters above for the robot Pioneer 3DX were already calculated in the dissertation of Martins [10], they are $\theta = [0.5338, 0.2509, -0.0134, 0.9560, -0.0843, 1.0590]^T$. His work also produced a Simulink [11] block diagram that generates the trajectory of differential robots following their parameter vector and the target points. The main blocks of this diagram are represented in Fig. 4.

The blocks presented in the Fig. 4 above, are in the following order. First, in the Reference Trajectories Block is defined the initial and final position, this block will also initiate the process and is being fed by the last position in the position array, controlling the robots path.

The next block is the Dynamic Model, its inputs are the Linear and Angular reference speeds, which are corrected by the dynamic equation with the robot parameters θ, resulting in the robot's estimated linear and angular velocities.

The last block is the Kinematic, the inputs for this block are the velocities generated by the previous block, and its output is the next trajectory point, with the location (x,y) and the orientation (ψ). The model of the Dynamic block is represented by (18), and for the Kinematic block by (19).

$$\begin{bmatrix} \dot{u} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} -\frac{\theta_4}{\theta_1} & \frac{\theta_3}{\theta_1}\omega \\ -\frac{\theta_5}{\theta_2}\omega & -\frac{\theta_6}{\theta_2} \end{bmatrix} \begin{bmatrix} u \\ \omega \end{bmatrix} + \begin{bmatrix} \frac{1}{\theta_1} & 0 \\ 0 & \frac{1}{\theta_2} \end{bmatrix} \begin{bmatrix} u_{ref} \\ \omega_{ref} \end{bmatrix} + \begin{bmatrix} \delta_u \\ \delta_\omega \end{bmatrix} \tag{18}$$

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} \cos\psi & -a\sin\psi \\ \sin\psi & a\cos\psi \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ \omega \end{bmatrix} + \begin{bmatrix} \delta_x \\ \delta_y \\ 0 \end{bmatrix} \tag{19}$$

The next step for the integration is the construction of a simulated environment, described by the next subtopic.

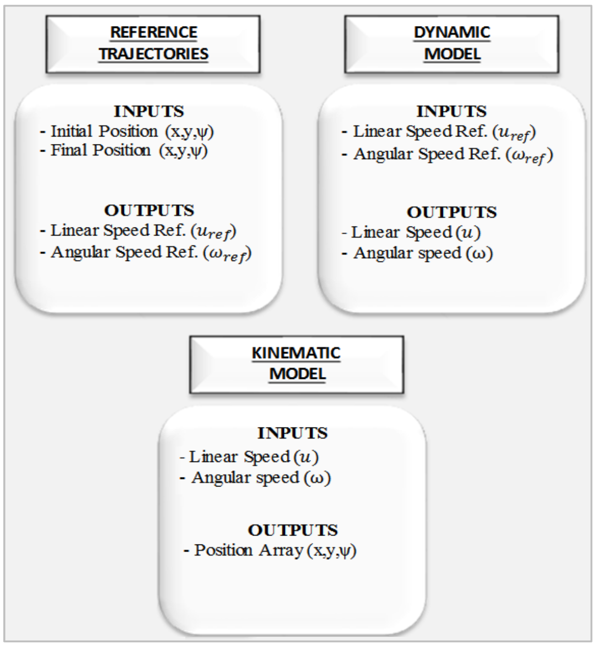


Fig. 4 Main blocks input/output

B. *Simulation Environment*

The simulation environment was developed in the computer graphics platform Blender 2.65 [5]. This software has a very large specter of applications, like the creation of simple 3D mesh, solid, textures, scripts for movement and others.

The purpose of the simulation can be split into two objectives, first, move the robot object following its dynamics, and second, render the images of two cameras positioned in front of the robot object, as shown in Fig. 5.

As the robot rendering will not be necessary in the applications using this simulation, its representation is very simple. Fig. 5 shows the cameras pointed to the ground, which is their default position, but their orientation can be easily changed depending on the objective of the application. This robot was included in two scenarios, the room and the office, each one with their characteristics focused on a specific objective. The first one, the room, is presented in Fig. 6.

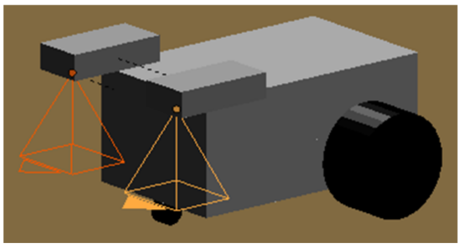


Fig. 5 Robot object representation

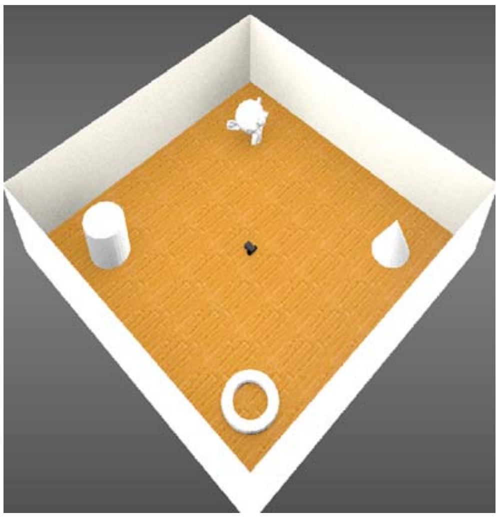


Fig. 6 Scenario: The room

This scenario is a 10x10 meter dimension and with 3 meter high walls. Even being quite simple, the scenario already has textures and two light sources generating shadows, allowing the production of 'good' quality images for processing.

The next one, the office, has exterior light interference, and complex object materials, like the mirror characteristics in the floor; its representation is shown in Fig. 7.

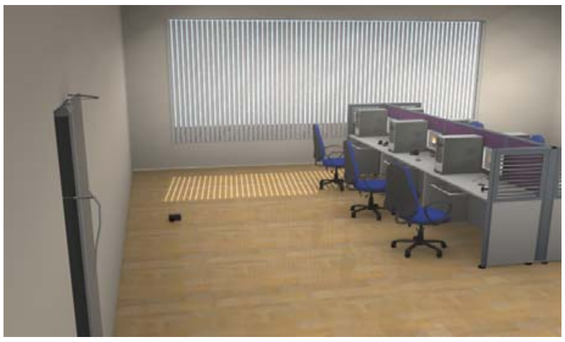


Fig. 7 Scenario: The office

Its dimensions are 12x9x3 meters, the high quality materials and the number of light sources make the image rendering of this environment very slow, being used when it is needed to test scripts with effects like reflection and inconstant luminosity. An example of a rendered image from the camera pointed to the front of the robot is shown in Fig. 8.

For the camera, the characteristics are easily changed depending on the simulation needs, but for the majority of uses, the values were focal distance = 32 mm, CCD 35 x 28.18 mm, and resolution 720x480. The output format was (*.jpeg), which can also be changed.

The last part of this project is the implementation of the dynamic and kinematic model in the developed environments, to simulate the robot's physics and generate an accurate approximation of its movement.

Fig. 8 Example of a rendered image from the robot camera

C. Implementing the Dynamic Model in the Computer Graphics Platform

At the simulation platform, each object created has an ID for their specification in each operation. As only one in which movement is needed, the differential robot has the most important ID, 'Cube_001', in this case. So all operations related to 'Cube_001' refers to the robot's movement.

All the calculations were made in the computer graphics platform 'Blender' [5], using the programming language python, and its module 'math', already available in the default downloaded software of the 2.65 version.

The first step is the initial and target points definition, so for one desired point, the main function '*Robot ()*' will run one time with the initial condition as the beginning and the target point as the end. For 'n' points, the main function will run 'n' times, just setting each time the initial condition with the previous point and the end with the target point.

The function Robot is responsible for calculating the next step of the robot until it reaches its target point with a defined tolerance, 1 cm tolerance of was used for the simulations and each step represents 0.033 seconds. This sample time was selected to make it possible for the generation of videos at 30 fps with the rendered images from the robot.

The max velocities, the robot measurements and parameters 'θ', are also specified in this function. For each target point it is necessary to specify a controller to drive the robot to the desired location. The controller's outputs are defined by (20) and (21).

$$\mathrm{u_{ref}} = tanh\left(errory * \frac{ky}{ly}\right) * ly * sin(\psi) + tanh\left(errorx * \frac{kx}{lx}\right) * lx * cos(\psi) \tag{20}$$

$$\psi_{\mathrm{ref}} = \left(\frac{1}{a}\right)\left(tanh\left(errory * \frac{ky}{ly}\right) * ly * cos(\psi) - tanh\left(errorx * \frac{kx}{lx}\right) * lx * sin(\psi)\right) \tag{21}$$

where, $errorx$ and $errory$ are given by the difference between the last and the current position of the robot, step by step. The values of the controller gains ky, kx, ly, lx were defined as 10, and a is the distance 'a' represented in Fig. 3. With the values of u_{ref} and ψ_{ref} it is possible to implement the equations of the dynamic and kinematic blocks. Thus, the simulation will run at 30 steps per second, and each step will return the location and orientation of the robot. With these data it is possible to execute the function Move in each step. In Blender, for example, the code below was implemented to move the robot, make the animation and render the images:

```
#Select the robot
bpy.data.objects['Cube_001'].select = True
#Get current position and rotation
cubepositionx = bpy.data.objects['Cube_001'].location[0]
cubepositiony = bpy.data.objects['Cube_001'].location[1]
cuberotation = bpy.data.objects['Cube_001'].rotation_euler[2]
#Get variations
x=xgl-cubepositionx;
y=ygl-cubepositiony;
r=rgl-cuberotation-pi/2;
#Move the robot
bpy.ops.transform.translate(value=(x,y,0))
bpy.ops.transform.rotate(value= r, axis=(0,0,1))
#Insert frames
bpy.ops.anim.keyframe_insert(type='Location',
confirm_success=True)
bpy.ops.anim.keyframe_insert(type='Rotation',
confirm_success=True)
#Render Images
Capture(frame)
```

For the code above, the green lines are just comments to help through the reading and the blue values are numbers or Boolean. The first step in the code is the selection of the robot, then it is possible to get its current location and rotation, and with these values we get the variation by subtracting the estimated (x, y, r) with the current ones.

With the variations, the robot can be moved and rotated, and, to visualize its followed path at the 3D view window, the location and rotation key frames are inserted. Fig. 9 shows the current position and rotation of the robot at the selected key frame, after the execution of the script.

The last line of the code runs the Capture function at the specific frame value. This function uses each camera ID to select, activate, render its image, and save at the specified directory. As the simulation has two cameras, the result will be two pictures each step, or 60 pic/sec.

The next topic will discuss the evaluation of the simulation, first related to the reproduction of the robot's movements, and then, the quality of the rendered images for use in image based applications.

III. Simulation and Results

As commented in the Section II-A, Martins [10] produced a block diagram in Simulink and made it available for download. Thus, it is possible to compare the results of the Blender simulation developed by the authors with the Simulink one. Fig. 10 represents the comparison between both trajectories.

The target points used for the generation of the trajectories represented in Fig. 10 are:

- Initial condition = (0,0);
- First point = (1,1);
- Second point = (0,1);

- Third point = (1,0);
- Last point = (2,2).

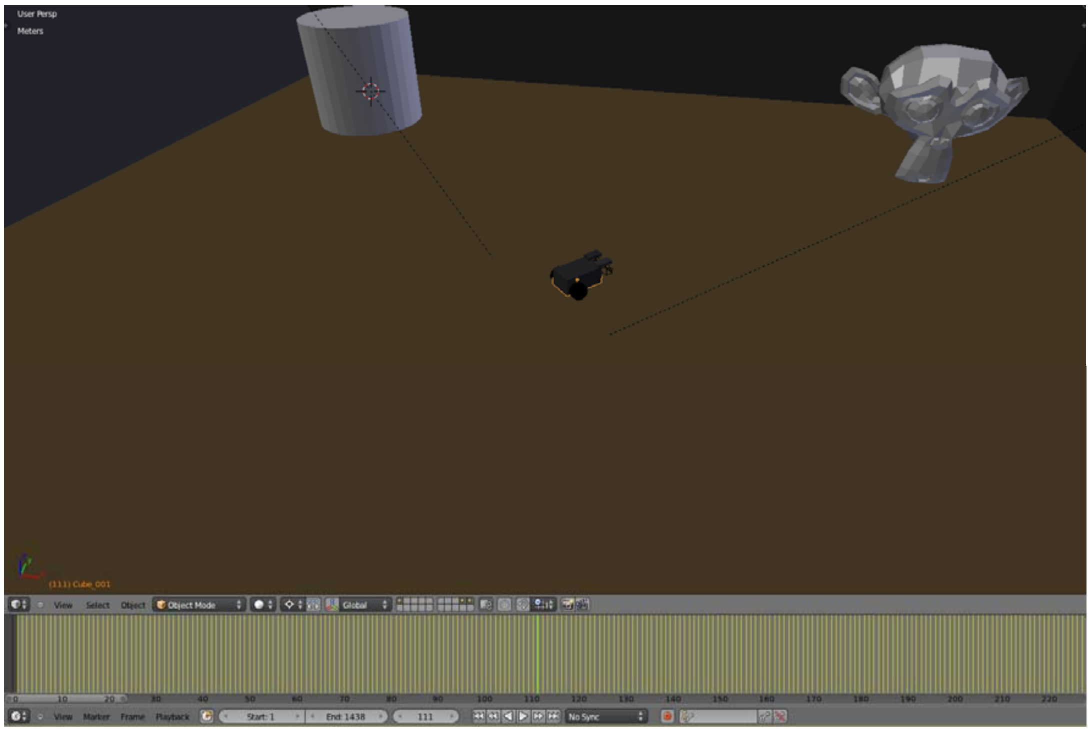


Fig. 9 3D view window

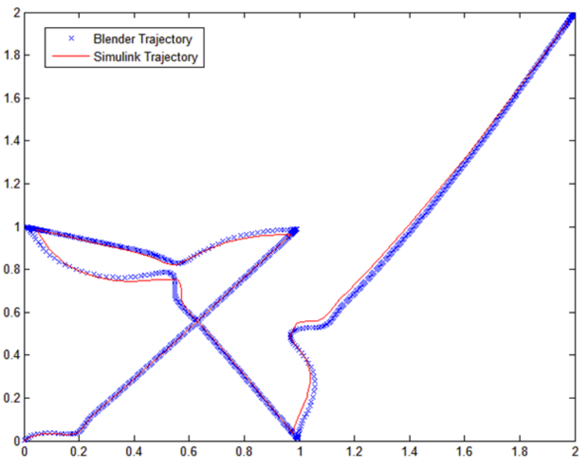


Fig. 10 Comparison between trajectories of the simulations in Blender and Simulink

Both trajectories follow the same behavior, but with some difference. Two explanations were found for that difference, the first one are the sample times used. In Blender, it was 0.033 seconds and in Simulink it was 0.1 seconds. The shorter sample time makes the Blender simulation produce more steps per second, and have a better response to rapid changes in robot position and orientation.

The second reason is the way that the simulations stop. In Blender, 1 cm of tolerance was specified, so when the difference between the position of both axis and the target point is less than that, the process stops. In Simulink that is not the procedure, it is necessary to specify the total simulation time, and the process of the robot's movement estimation will continue until that time ends, resulting, sometimes, in a very large amount of steps very close to the target point.

Another comparison is the error of both simulations from the target point [0,1], as it is the second point in the specified path, the minimum value in the graphs represents the robot passing through the point. Fig. 11 shows first the error in 'x' axis, and second 'y' axis, both errors vs. the normalized steps.

In Fig. 11, Blender simulation obtained smaller minimum errors, $(7 * 10^{-4}, 1.3 * 10^{-3})$, in comparison to Simulink, $(4.52 * 10^{-2},\ 3.3 * 10^{-3})$. Also, the MATLAB simulation seems slower for all the target points but the last one, the motive is that, the Simulink block diagram generates a large number of steps at the end of its execution, so the cut of some steps may cause this effect for a normalized representation. The motive for these extra steps at the end is because the Simulink block diagram is based on time and not in a tolerance from the target point.

The final result (Fig. 12) is an application for rendered images validation that consists of the estimate of the path followed by the robot using its camera pictures saved in each step using Lucas-Kanade method to estimate the variation of pixels between subsequent images, and epipolar geometry to calculate the robot's translation and rotation.

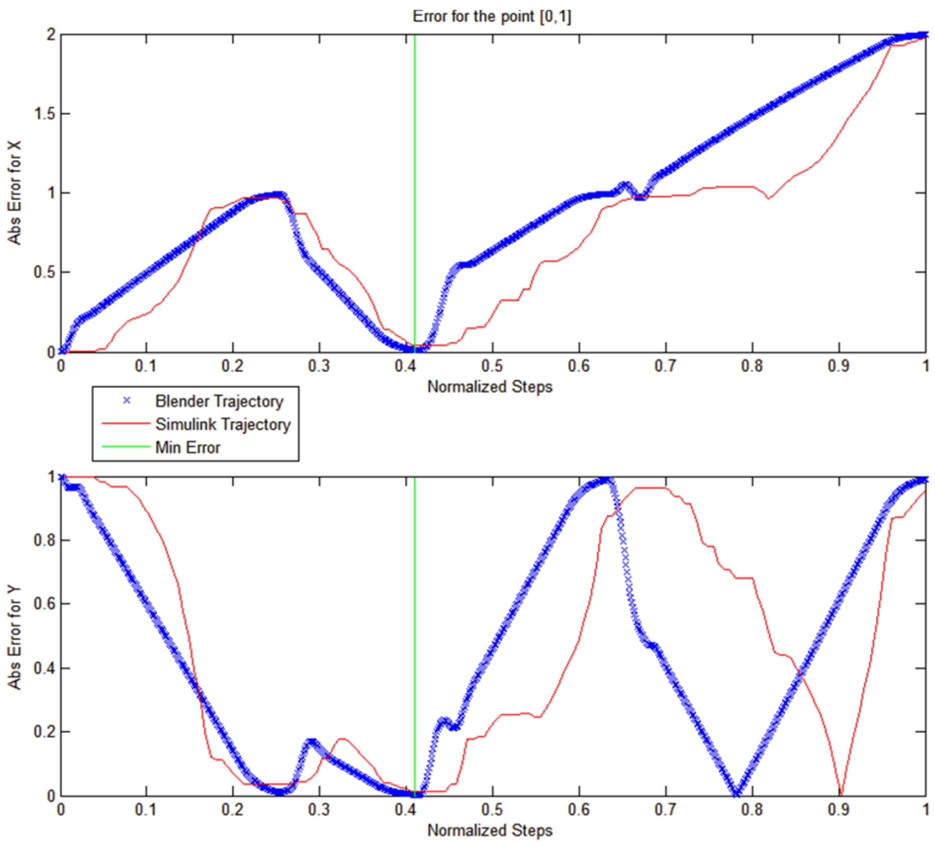


Fig. 11 Comparison between trajectories of the simulations in Blender and Simulink

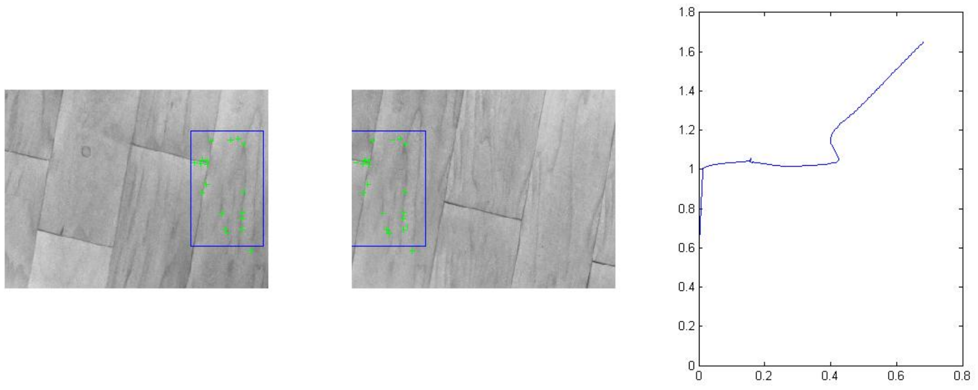


Fig. 12 Image based trajectory estimation using simulation results

.

This application consists of the use of images from both cameras pointed at a textured floor to track the pixels of subsequent pictures, as well as the use of visual odometry to estimate the robot's movement with the related pixel localizations. The path for the estimation was (0,0) to (0,1), (0,1) to (1,1), and (1,1) to (2,2).

IV. Conclusion

The main objective of this paper is to exemplify a methodology for enabling the use of computer graphics platforms in the field of mobile robotic, focused in image-based navigation algorithms. This same process can be duplicated for dynamic and kinematic models of different robots, or for another platform, by making just a few changes.

Thus, the major applications of this work are aimed at its duplication and implementation in a very large field of image analysis, and its methodology is particularly easy to adapt for the field of visual navigation.

Even though at this point the research can be very useful, some points still need improvement, such as the creation of a library with different scenarios ready to use, the modeling of more sensors and actuators, and use the Blender game engine

API to interact with the robot.

REFERENCES

[1] D. Jonassen. “O uso de novas tecnologias na educação a distância e a aprendizagem construtivista” (*The use of new technologies in distance education and constructive learning*). In open, Brasília, 2016, n.70, apr/jun, 1996. pp 88.

[2] Gazebosim. Gazebo. http://www.gazebosim.org/. Retrieved October 15, 2015.

[3] Cyberbotics. Webots. http://www.cyberbotics.com/. Retrieved October 15, 2015.

[4] Coppelia. V-rep. http://www.coppeliarobotics.com/. Retrieved October 18, 2015.

[5] Blender 2.65a. blender.org. October 26, 2016. Retrieved October 26, 2015.

[6] M. Sharifi, X. Chen and C. G. Pretty, "Experimental study on using visual odometry for navigation in outdoor GPS-denied environments," 2016 12th IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications (MESA), Auckland, 2016, pp. 1-5.

[7] Y. Liu *et al.*, "Stereo Visual-Inertial Odometry With Multiple Kalman Filters Ensemble," in IEEE Transactions on Industrial Electronics, vol. 63, no. 10, pp. 6205-6216, Oct. 2016.

[8] De La Cruz, C.; Carelli, R. “Dynamic modeling and centralized formation control of mobile robots”. In: 32nd IEEE Conference on Industrial Electronics. (S.l.: s.n.), 2006. p. 3880–3885.

[9] Zhang, Y. *et al.* “Dynamic model based robust tracking control of a differentially steered wheeled mobile robot”. American Control Conference, v. 2, 1998.

[10] Martins, F. N.; Carelli, R.; Sarcinelli-Filho, M.; Bastos-Filho, T. F. “Dynamic Modeling and Adaptive Dynamic Compensation for Unicycle-Like Mobile Robots”. 14th International Conference on Advanced Robotics - ICAR 2009, Germany, June, 22-26, 2009.

[11] © 2016 The MathWorks, Inc. MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.