# Simulation of Stewart platform for lunar-martian rover simulator

Michał Barański
Department of electronics,
telecommunications and informatics
Gdańsk university of technology
Gdańsk, Poland

***Abstract—***

***Keywords—Stewart platform, simulation***

## I. Introduction

W przygotowaniu...

## II. Background

## III. Stewart platform simulation

The selected motion platform (MotionSystems PS-6TM-150) is a six degree of freedom Stewart Platform. Unlike typical Stewart Platforms that are driven by linear motors or hydraulic actuators, this particular model is driven by six regular rotary motors, through a high gear ratio gearbox that are hidden in all 3 sides of the platform. This allows the platform to have a compact design while maintaining high load requirements (the platform can handle up to 150kg of load).


Figure 1. MotionSystems PS-6TM-150 Stewart platform [2].

The arms connecting top and bottom parts of the platform are configured with a universal joint on the top and a spherical joint on the bottom that connects to an additional short arm fixed to the gearbox output shaft. This arm arrangement gives the platform an impressive movement range with 52° pitch, 50,6° roll, 45° yaw angles and 22cm surge (front-back), 20cm sway (left-right) and 23cm heave (up-down) linear movements. The platform has a solid, stable base, supported by 3 adjustable legs and 3 wheels for transportation, with an additional option of anchoring the platform to the ground by using pre-installed mounting holes. This design forms a professional motion platform with very high movement repeatability and impressive performance, that is also safe for the user with safety integrity level up to 3. [1] The software required to control the platform (ForceSeatPM) is also a MotionSystems product and is included with the platform. It has built in integrations for most popular simulators and games (e.g. VBS3/4, Microsoft Flight Simulator 2020, X-Plane12, Matlab/Simulink and many more [3]). Additionally it can do hardware diagnostics, monitor platform movements and compensate for head movement in VR applications (through VR HeadWay plugin). It also supports development of custom applications through an SDK (ForceSeatMI/ForceSeatDI, available separately) that supports Unity, Unreal Engine, Matlab/Simulink, as well as native C/C++/C# and Python. [2]

Simulation was implemented using multibody physics provided by Bullet Physics Engine. “Bullet solves the equations of motion for articulated rigid bodies in generalized coordinates while simultaneously satisfying physical constraints including contact, joint limits and actuator models.” [4] Multibody physics, in contrast to rigidbody physics, offers fully stiff joints between elements by simulating only those degrees of freedom that are needed for particular joints between physics elements. This advantage however comes at a cost, that the structure of connected elements must form a tree (it cannot have cycles). To overcome this limitation the physics model was assembled from two separate multibody trees, connected together using regular rigidbody physics joints. The first tree consists of the base of the platform along with all six motor shafts with the short arm that are connected to the base using a revolute joint. Those shafts are driven by simulated motors, that allow control of current angle of rotation by setting a position target in degrees. The motor simulation tries to reach that position, within a limit defined by a maximum impulse. The second multibody tree used in the simulation consists of the top part of the platform and all its arms, connected to the top by a universal joint created by using two revolute joints

and an additional multibody node between the top and the arm. Both multibody trees are then connected together at the spherical joints on the bottom of the arms, using a point to point constraint.

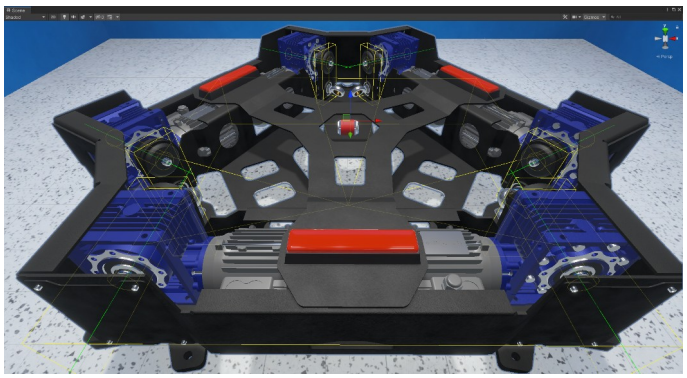

Figure 2. Bottom multibody tree.

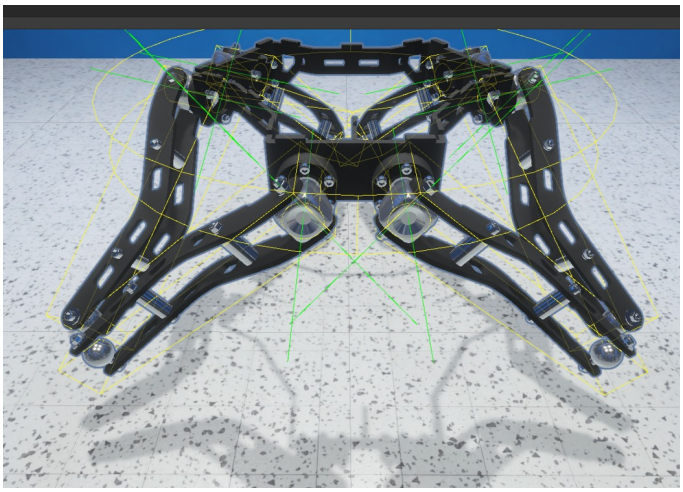

Figure 3. Top multibody tree.

Both multibody trees are shown on figures 2 and 3. The yellow frames represent a multibody node while the green lines show the rotation axes of revolute joints between the nodes. The yellow lines show connections between objects.

A physics model prepared in this way behaves just like the real platform – it is controlled by controlling the angles of rotation of each motor. The physics engine then calculates positions and rotations of each element based on the defined constraints with a constant time step of 5ms (200Hz), using Runge-Kutta (RK4) integration for additional accuracy.

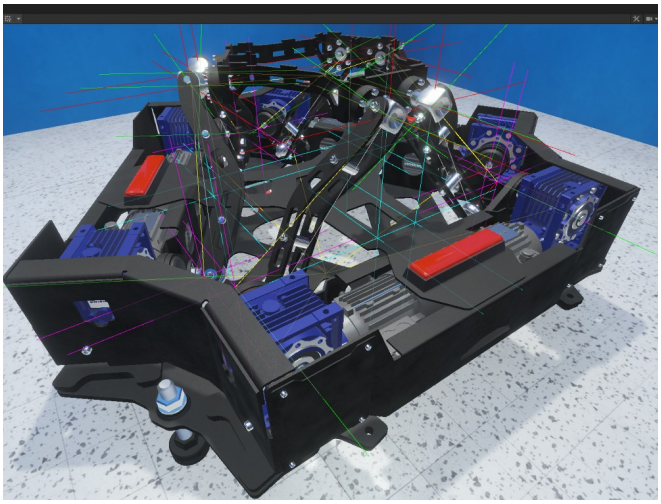

Figure 4. Control points.

To control this physics model it is necessary to use the same methods that are used in real Stewart platforms. The control algorithm has to calculate desired rotations of all six motors for an input consisting of top platform position and rotation. One such method is inverse kinematics analysis. “It is simple to solve compared to forward kinematics and can be used to plan a specified trajectory that uses leg lengths by providing any specified position of the centre of the platform top plate”. [5] The selected method controls the platform by performing inverse kinematics to calculate desired lengths of each arm, based on arm origin points on the bottom platform (not moving) and end points on the top platform (rotated and translated with the desired input). The arm lengths are then used to calculate the desired motor rotations. This process is illustrated on figure 4. The red points show the arm end points on the top platform, magenta coloured points are the arm origin points and yellow lines represent the arms. Additionally cyan points show the arm end points projected onto the plane of the platform base, that are used internally in the inverse kinematics process. A complete model of the platform is shown on figure 5. In addition to previous figures it shows the local coordinate systems of each motor shaft as red, green and blue lines originating from each shaft, and a complete graphical model.

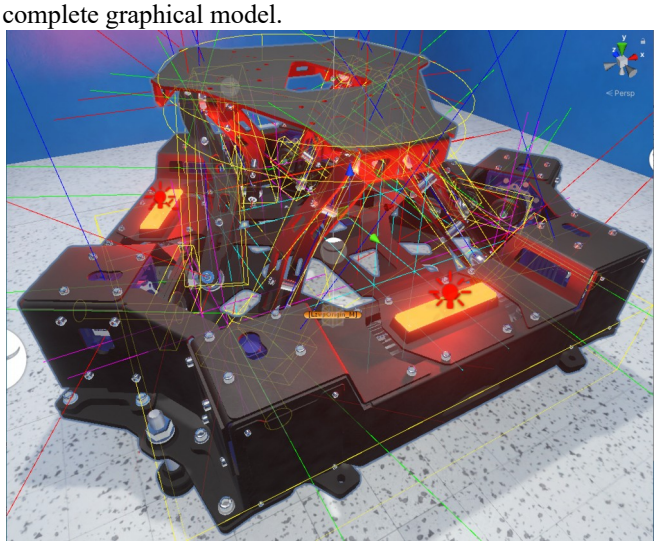

Figure 5. Final platform model.

The last essential part of the platform simulator was a method of transmitting the control parameters from the rover simulation. The lunar-martian rover simulator provides a telemetry API that can be used to query current rover simulation parameters using UDP over network, by using their string paths. The API also supports setting up a persistent cyclic read where the rover simulator itself sends the requested values with specified send rate. This mode is more efficient than querying for values because it sends only a short id number instead of a full string path, and also provides lower latency, therefore it was chosen for handling the communication between simulators. The data that are transferred consist of 3 float values representing the pitch, roll and yaw angles, and are available under “vehicle/position/pitch”, “vehicle/position/roll” and “vehicle/position/yaw” paths in the rover simulator telemetry API. The data rate was configured at 30 sends per second, so the rover simulator sends one data packet every 33ms, that consists of the 3 selected values, their id numbers and the packet header, which in total make up 42 Bytes of data. As soon as this packet arrives, the rotation angles are written to a data structure shared with the main application thread (the network communication runs as an asynchronous task), and then are sent to the platform control algorithm in the

application update loop. In addition to sending simulation parameters, the telemetry API has an integrated method of measuring latency between the connected applications. Both applications, at any time can send a ping request packet and the other side will respond with a response packet, allowing for measurement of the round trip latency. This method is used in all latency tests.

To sum up, the process of creating a simulated counterpart of a mechanical device used in this project consists of several steps. First of all, it is essential to have an accurate 3-dimensional model of the device, that includes all moving parts. This model can be created in 3d modelling software based on the device's technical drawings (as done in this project), or exported directly from CAD software if access to such data is available. Any errors in parts dimensions made at this stage will directly influence the physics simulation, so care must be taken to minimize them. The second step is to create a physics model, using the obtained graphical model. This involves creating one or more multibody trees with one node per each moving part of the device and defining constraints that join the parts together and/or limit their movement. Proper graphical model preparation makes this step much faster – it should be prepared in a form of an object tree where objects directly influencing each other (e.g. joined by a rotational or translational joints) are arranged in a parent/child relation. Furthermore, placing rotary objects origins in the axis of their rotation at modelling stage eliminates the need to set up joint offsets during joint configuration.

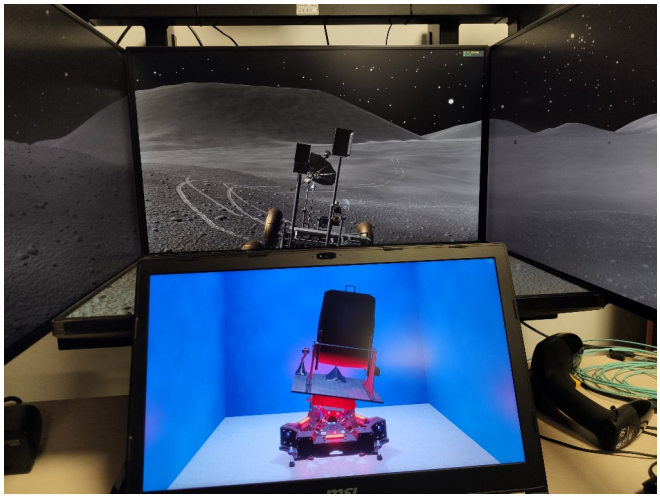

Figure 6. End result.

The last step is to implement software needed for the machine to function correctly. A well defined physics model will very closely match the real device's behaviour so it is necessary to use the same control methods as are used in the real device. In this project the software was divided into two parts – a low level component that directly drives the motors, based on the desired platform position and rotation, and a high level component that receives telemetry data from the rover simulator and uses the low level driver to control the platform. The end result is shown on figure 6 which shows the platform simulator running on a laptop, connected to the lunar rover simulation running on a mini CAVE in the Immersive 3D Visualization Lab.

## IV. Results

W przygotowaniu...

## V. Conclusions

## References


[1] *PS-6TM-150 Product card - https://motionsystems.pl/wp/wp-content/uploads/2020/08/MotionSystems_ProductCard_PS-6TM-150.pdf*

[2] *MotionSystems PS-6TM-150 - https://motionsystems.pl/product/6dof-pl/ps-6tm-150/*

[3] *List of supported aplications - https://motionsystems.pl/supported-games/*

[4] *Yie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez and Vincent Vanhoucke, Sim-to-Real: Learning Agile Locomotion For Quadruped Robots, arXiv:1804.10332*

[5] *Berkay Volkaner, S. Numan Sozen, V. Emre Omurlu, Realization of a Desktop Flight Simulation System for Motion-Cueing Studies*