Contents

1 Introduction

This paper contains a study of simulation of autonomous helicopters and service robots that has been performed in the framework of the WITAS [15] project. That project concerns research in the area of autonomous aircraft and other autonomous systems. One of major objectives of the project is to investigate tasks that unmanned autonomous aircraft will be able to solve in future. For this purpose a virtual environment is needed to simulate these tasks and solutions. The environment includes a command and control architecture with a system of active vision. One of the tasks for an unmanned autonomous aircraft, e.g. a helicopter, could be to monitor traffic situations as well as providing emergency help. The tasks of the robots include servicing the helicopters, as well as loading and unloading various objects to and from the helicopter.

The most complex part of the project includes design of control systems working in several layers. Note that our description below covers just a little fragment of the WITAS software systems and related research activities, which can be found in [15]. The control system for the autonomous vehicle is organized as software tools in three layers (see Figure 1):

- A deliberative layer produces plans, e.g. a plan of movement.
- A reactive layer produces responses when certain events happen.

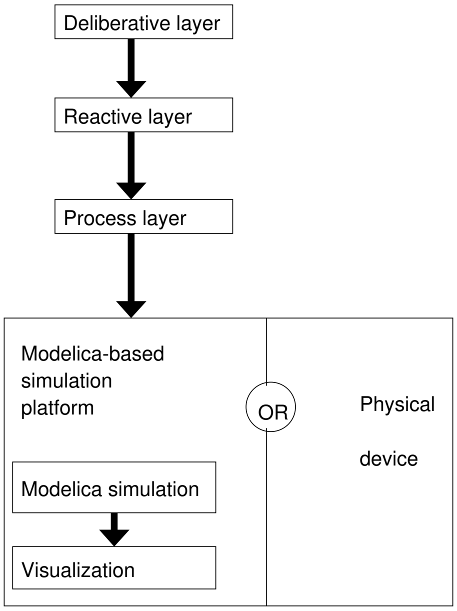

Figure 1: Layers of a simulation architecture for autonomous vehicles.

- A process layer gets information from sensors and produces signals for mechanism actuators (e.g. motors).

The mechanical model is placed ”under” these layers. Depending on the environment either a mathematical model of the mechanism is linked to the control system software, or a real physical device in the laboratory or on the field is used as a mechanical model. In order to test and verify the control system the mathematical model and the real device should respond to actuators in the same way and give the sensors the same answer.

Therefore, it is necessary to enhance the realism of simulations in order to verify the correctness of the control systems.

This can be done in different ways; in the case of flying and moving mechanisms it is quite important to take the dynamics of the mechanisms into account. Initially the models were represented as point masses. Since the dynamics can be simulated in Modelica[7] we designed the corresponding models of the helicopter and the robot in this language.

The assemblies contain many components that move and rotate in certain ways. This information has been extracted [5, 4] from CAD models designed in the SolidWorks tool [14] and glued with other Modelica code. Simulation has been performed using the Dymola tool [2], the multibody system library[6], together with online and offline visualization and interactive control[4].

The report contains a description of the helicopter model (Section 2) and the robot model (Section 3). Conclusions are given at the end of these sections.

2 Helicopter modeling

The helicopter model consists of two major parts:

- A control system [13] written in Ada (later translated to C). This control system has been initially tuned with a simplified helicopter model [11, 12] described using explicit motion equations.
- A mechanical dynamic system modeled in Modelica where the major part of the code is generated automatically by the SolidWorks–to–Modelica translator.

These parts communicate with each other via function calls.

2.1 The control system

The part of the control system which communicates with the Modelica model is the process layer. This system takes the plan (helicopter mission) as commands expressed in a language FCL (Flight Command Language). Typical commands in this language look like `FLY-TO- POINT(x,y,z,v,p)` or `LAND()` where `(x,y,z)` are space coordinates of a mission point, `v` is the required velocity, `p` is the required precision. The command `FLY-TO-POINT` means that the control system will try to steer the vehicle to a specified point so that the magnitude of velocity is `v`. Each command is considered as executed as soon as the vehicle occurs at a distance less than `p` from the target point. Then the next command is requested from the upper layer of the control architecture. For testing purposes the test sequence of FCL commands is just read from an input file.

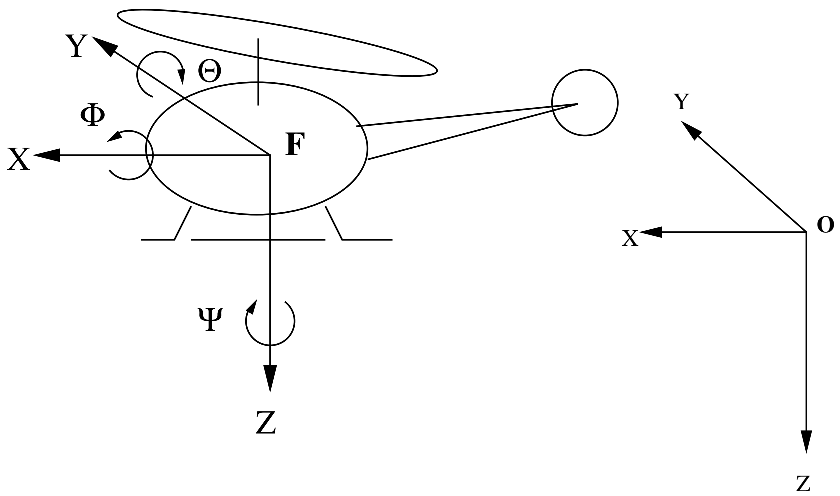

Figure 2: The origin (O) frame and the body-fixed (F) frame.

In order to describe the architecture of the control system, the coordinate frames (Figure 2) should be considered. The origin coordinate frame O corresponds to the ground, inertial system. The body-fixed coordinate system, F is attached to the main part of the helicopter. The relation between these systems is determined by the position of the helicopter and its rotation around each axis. The rotation angles Ψ, Φ and Θ are yaw, roll and pitch rotations around the Z, X and Y axes correspondingly.

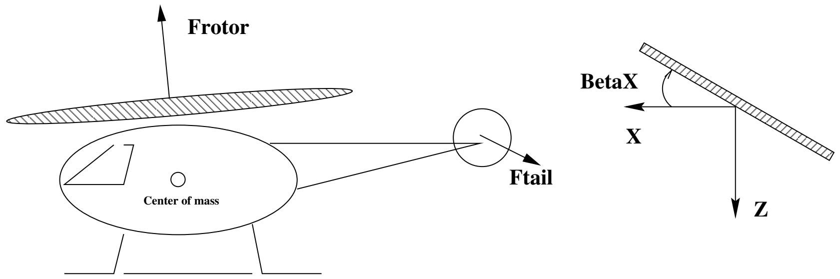

Figure 3: Positions of the forces F_{rotor} and F_{tail} on the helicopter. The angle β_x is the deviation of the rotor plane from the X axis.

The magnitude of two forces (F_{rotor} and F_{tail}) and values of two angles (β_x and β_y) are delivered to the helicopter model by the control system (see Figure 3). The force F_{rotor} is a lifting force applied to the main rotor of the helicopter. The rotor is located above the center of mass of the helicopter. The rotor is tilted to the angle β_x around the local Y axis and angle β_y around the local X axis. Finally, F_{tail} is the force created by the tail rotor.

The control system switches between different $modes$. A switch happens when the execution of a new command starts, or the helicopter position or rotation reaches a certain value (i.e. reaches a threshold value and cause an internal event). Each mode has a specific feedback loop which maps current position, velocity and rotation to new force magnitudes and angle values. Rules defining these modes are described in [13, 8].

2.2 Mechanical model of the helicopter

The mechanical parts of the helicopter model (see Figure 4) has been designed in SolidWorks [14]. There are eleven parts in the model. The kinematic skeleton for the moving parts of the model is shown in Figure 5. The Modelica connection diagram of the helicopter is shown in Figure 6.

The major parts are the main body of the helicopter, the main rotor, the tail rotor, left and right landing gears, and the door. There exist minor parts, which were convenient to define for debugging of the model and integrating with the robot model. The ”reference cylinder” and load platform are attached to the main body. Three intermediate parts used for rotation are inserted between the main body and the rotor. All the parts are considered as rigid bodies. They are connected by rigid, revolute (rotational) and prismatic (translational) joints.

The size of the helicopter used in the project is $1.5 \times 0.5 \times 0.5$ m. The weight is about 50 kg.

The mechanical part of the helicopter is just one of components in the helicopter model. There exists a set of components which cannot be defined just by drawing in the CAD tool. In Modelica these components are connected to the mechanical part by ten `MbsCutB` connectors.

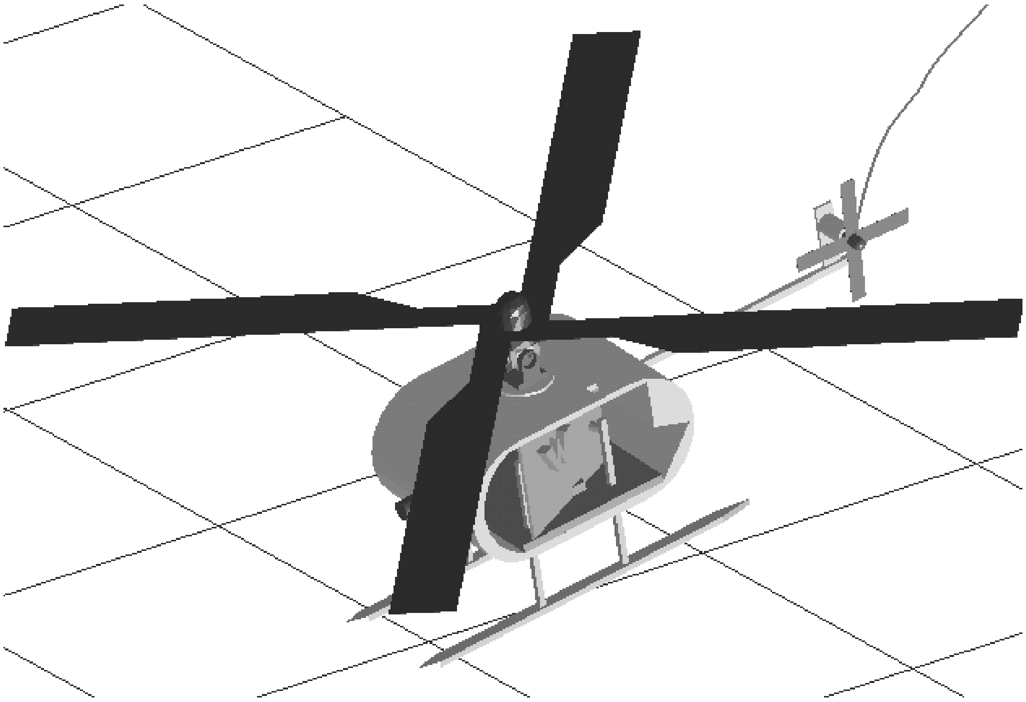

Figure 4: Helicopter model designed in SolidWorks.

This connector is a data structure which contains position and rotation of a coordinate frame attached to the mechanical part in question at some point. In addition, this connector contains variables defining the force and the torque that can be applied to this point. They are denoted as pentagons on the leaves of the Modelica connection diagram (Figure 6)

The components used in the model are the following:

Springs in the landing gears. A spring and a damper is inserted between the landing gear and the helicopter. This models the flexibility of the landing gear necessary for soft landing.

Motors for rotating the rotors and the door. These motors have a reference angle which changes with constant or varying velocity. A feedback torque is applied to the main and tail rotors according to the current and desired angular position and velocity:

$$t = k(q_{ref} - q) - d\dot{q};$$

where q is the current angle, q_{ref} is the reference angle, k and d are feedback coefficients, t is the resulting torque.

Cabin floor contacts. Four `MbsCutB` connectors are placed on the floor of the helicopter. These can be used in order to attach a load to the helicopter. If a load has the corresponding four `MbsCutB` connectors, some attraction force is set up between the points at the helicopter floor and the points at the load.

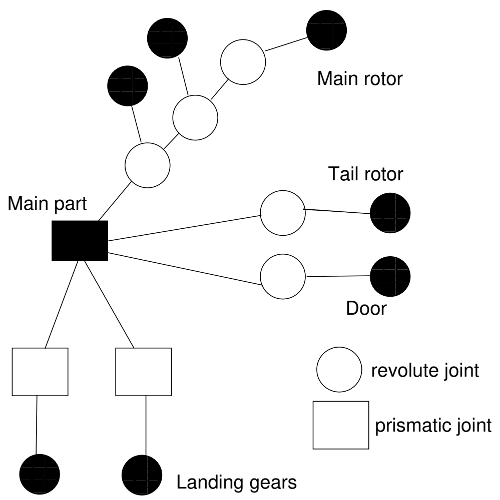

Figure 5: Kinematic skeleton of the helicopter. Five revolute joints, two prismatic joints and one rigid joint are used.

Ground contacts. There are four ground contact point specified for landing gears. There exist two points (front and rear) for each gear. A force is created as soon as a contact point is close enough to the ground (otherwise it is zero). The force is defined as a sum of horizontal friction force and vertical collision force.

$$\begin{aligned} F &= -(f_1 S_b' n_y) + (-f_2 S_b' n_x) + (-f_3 S_b' n_z) \\ f_1 &= c(s - s_0) + d v_y \\ f_2 &= c_{fric} v_x \\ f_3 &= c_{fric} v_z \end{aligned}$$

where

- S_b is the current rotation matrix of the contact connector on the landing gear.
- n_x, n_y, n_z are unit vectors.
- (v_x, v_y, v_z) is the current velocity of the contact connector relative to the ground.
- s is the current height of the landing gear contact point relative to the origin.
- s_0 is the height of the ground level relative to the origin. It is a constant if a flat ground is modeled. Otherwise it is a value defined externally as function $s_0(x, z)$.
- c and d are spring and damping coefficients for collision with the ground. We use here a simplified force-based collision model. More advanced models have been designed later, see [3].
- c_{fric} is the coefficient of the friction constant preventing the helicopter from sliding after landing.

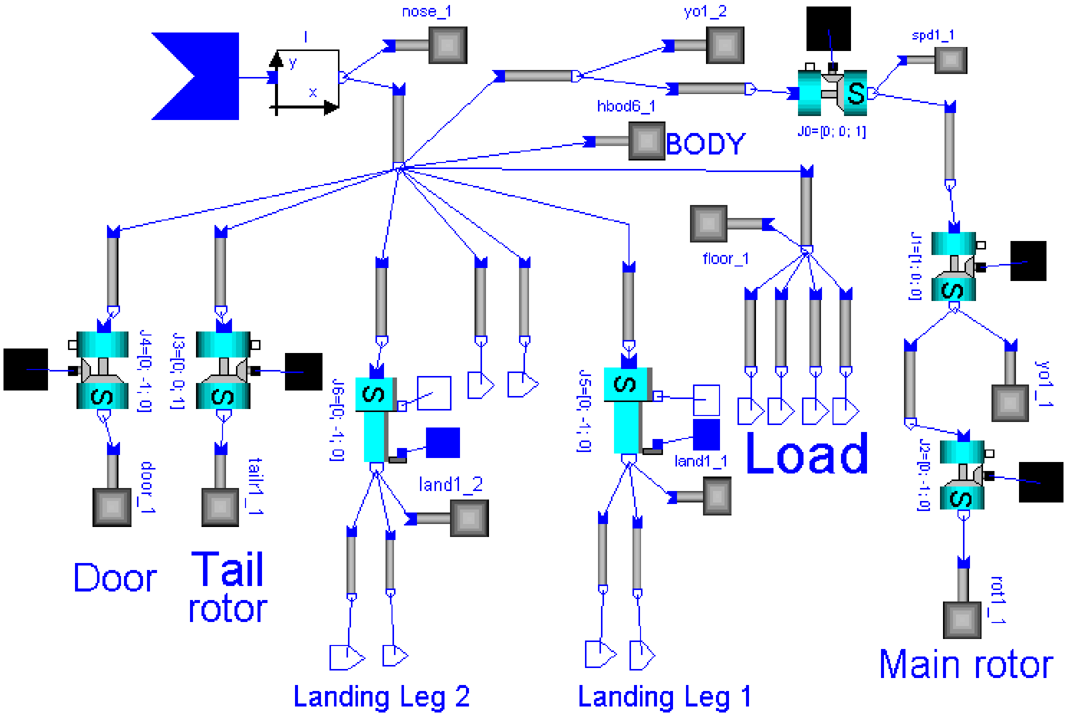

Figure 6: Modelica connection diagram of the mechanical part of the helicopter.

Just after a landing, if four such forces are applied to the landing gears of the helicopter, these forces compensate the gravity force and the helicopter stays on the ground. Guessing the right values of the coefficients is a difficult tasks. This has been done by trial and error. In order to achieve naturally looking movements during the landing: $c = 10^5, d = 10^3, c_{fric} = 15$.

There are some other components that can be easily attached to the model

Load. A load of appropriate size can be attached to the floor of the helicopter. The force between the load and the floor is modeled by stiff springs.

Ground level height. The height of the ground level can be defined by a table or by an external function which contains the geometry of the ground. This gives the possibility to model a landing failure if the landing point is on a slope. However if a ”mountain” appears between two mission points, the control mechanism cannot prevent collision, since there is no built-in collision avoidance model.

Wind. Additional constant, random or customized forces and torques in arbitrary directions can be applied in order to destabilize the helicopter. This way wind and turbulent air flows can be modeled.

2.3 The Integration of the Helicopter Control System and the Helicopter Mechanical System

The mechanical system (the Modelica model) and the control system (the C or Ada model) have their own loops for finding solutions of arising mechanical and control equations. The control system uses an Euler method and fixed time steps (0.01 seconds).

The mathematically correct way of integration of Modelica model with sampled, discontinuous, external signal is using a `when sample(...)` statement. This statement triggers an event, updates the variables, and restarts integration each time an event occurs. Unfortunately, integration restart takes too much time, and computation time will be non-realistically large in this case. However, the control system is designed so that its output values change smoothly. Therefore we use external functions directly.

In order to synchronize the solvers special interface routines have been developed. One iteration of the control system loop might correspond to many iterations of the mechanical system loop. The time in the control system increments by a fixed value at every solver step. The time values used in the solver steps in the mechanical system are dependent on the used solver. We found that an adaptive solver should be used, since other solvers lead to simulation failure (see below). Therefore, the time steps might vary. Simulation code generated by Dymola tool for the Modelica model of helicopter contains a number of solver iterations. During each iteration the external functions are called as follows:

- A solver step starts.
- Modelica simulation calls an external function that obtains the most recently saved values for forces and angles. These values were saved in the memory buffer during one of the previous steps.
- According to the equations of mechanics, the current helicopter acceleration, velocity and position is determined.
- The Modelica simulation calls an external function with the current helicopter position, the current time and other variables as parameters.
- If the modeled time in Modelica simulation code is larger than or equal the modeled time in the control system, the control system takes the parameters of the helicopter and evaluates new forces and angles that should be applied to the helicopter. New forces and angles are stored in the memory buffer. Also, the control system advances its time.

It is also possible to move the first call of the external function to the end. Since Modelica language specification does not explicitly define the order of evaluation of external functions, this order can be controlled indirectly by adding artificial dependencies between function calls. In this case the compiler attempts to reorder the sequence of the calls. For instance if two equations $a = f_1(b)$ and $c = f_2(a)$ are specified, and b is a known variable, f_1 will be called first. This is not a safe way, however, and should be used with care.

Additional uncertainty arises since functions in Modelica can be called many times at different solver iteration steps with "trial" time which can go forward and backward. In our simulations we made a simplification and used the largest reached time value and ignored all new simulated time values which were less than the largest one.

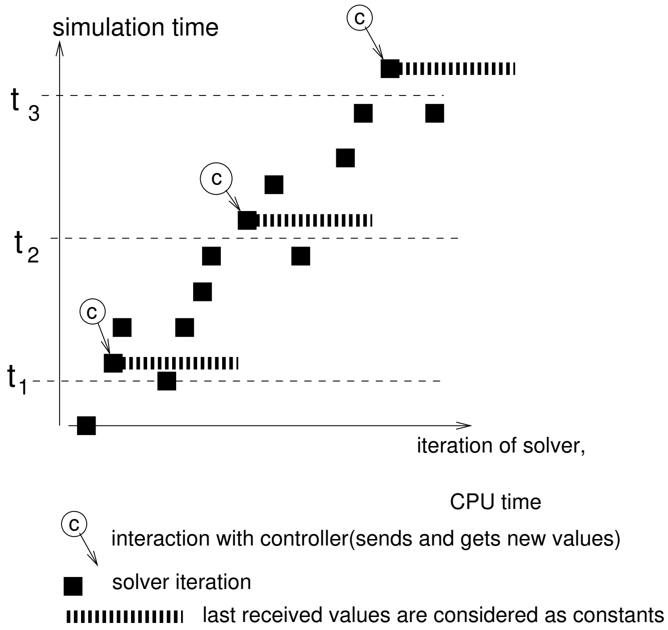

Figure 7: An approach to sampling values at regular simulated time intervals from an external source. The solver iterations are represented as small black boxes. As soon as the time corresponding to the iteration is larger than t_1 (the same applies to t_2, t_3, etc.), a value is read from the external source.

In the solver the external function for retrieving the data from the controller is called at each iteration of the solver, and different values of $time$ are given to this function. As soon as time becomes larger than the communication instant, e.g. t_1 at iteration s_1, the controller is called and returns values (forces and angles). These values either remain constant until the iteration s_2, or smoothly change (using one of several smoothing approaches) as described below.

The solvers of ordinary differential equations, in particular those used with Modelica simulation code, require that all external functions during continuous integration are differentiable up to n-th order (where $n \geq 1$ is the order of the numerical integration method used in the solver).

Therefore all input data should be smoothed using one or several splines connected together. Three variants of smoothing were tried: initial discontinuous signal (Figure 8(a)), continuous (Figure 8(b)), and signal with continuous first derivative (Figure 8(c)). The last one is constructed by gluing

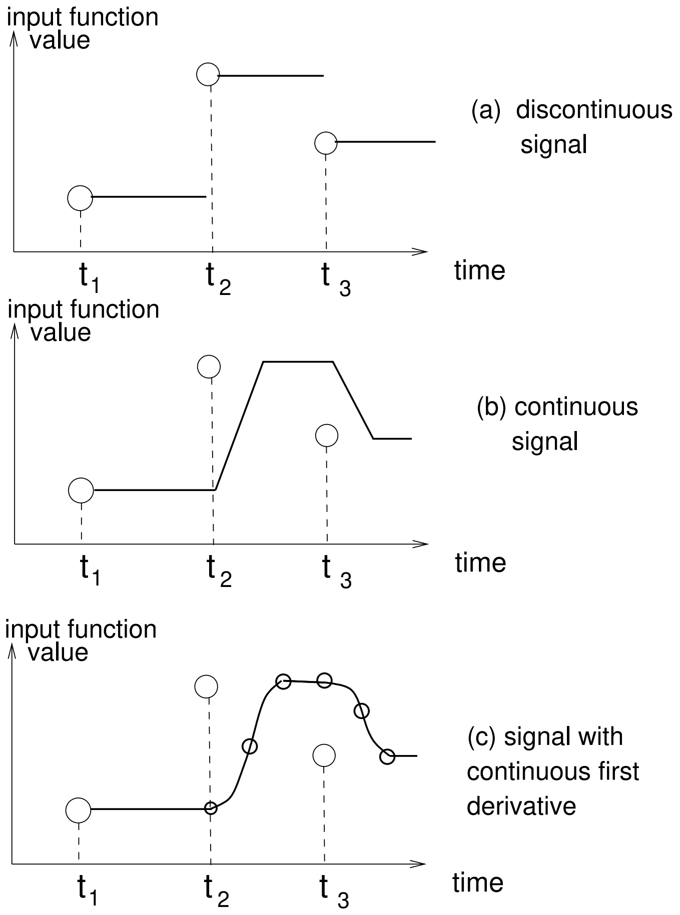

Figure 8: Three variants of smoothing data in an external function. The circles represent the values obtained at sampling times t_1, t_2, t_3. Since an external function should not be discontinuous, one of three smoothing algorithms is performed.

together two second order polynomials and one constant interval. Despite of these variations the numerical results and simulation performance were roughly the same in all three cases. Therefore we made a conclusion that a discontinuity of the input data does not affect the solver[1] substantially in our model.

We have a hypothesis regarding an accurate approach to the co-simulation of the control system and the mechanical system. If the control system uses a fixed-step explicit equation solver, the mechanical system should use the same, fixed-step explicit equation solver, with probably smaller step size. In practice, however, the mechanical system is quite complicated, therefore the time step for it should be extremely small in order to use an explicit method and obtain a solution that converges. Computation time will be non-realistically large in this case.

[1]We use the DASSL solver[10] implementation in the Dymola tool[2].

2.4 Helicopter Visualization

The helicopter visualization is obtained by the MVIS tool. The following properties were particularly useful for visualization:

The mission lines. The points used in the flight specifications in the Flight Command Language (mission points) are shown as boxes in space, connected by straight lines. Helicopter normally should fly along the lines and turn near the boxes. It is easy to see cases when the flight deviates from the mission lines.

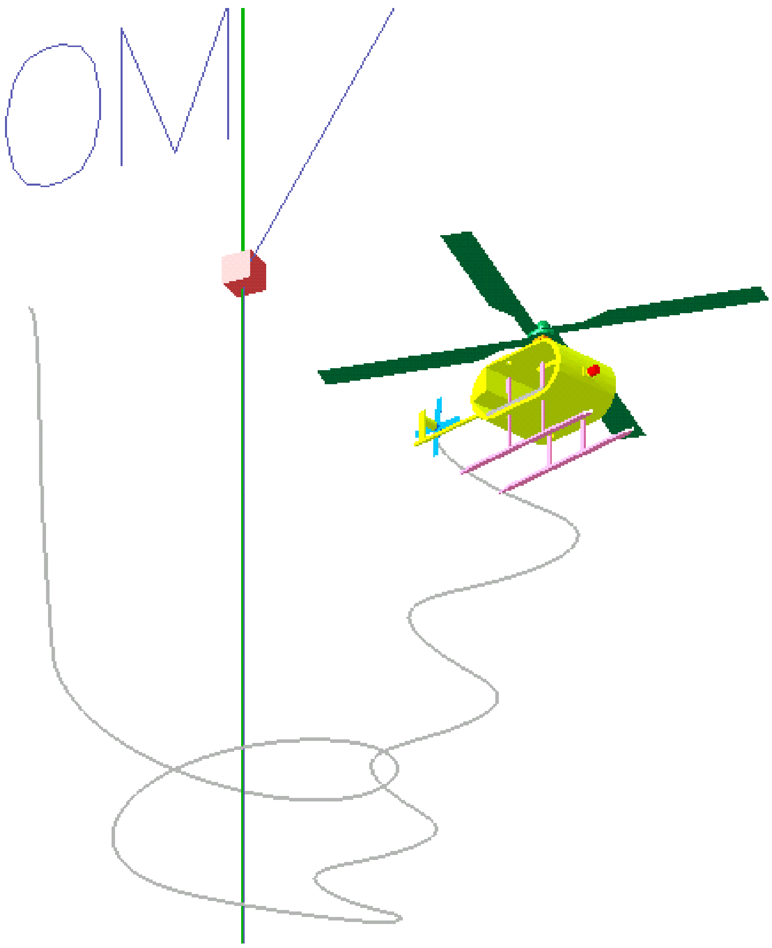

Figure 9: Trajectory of the tail rotor of the helicopter.

Helicopter trajectory. The trajectory of some part of helicopter is shown on the screen and compared with the mission line. If the main part of the helicopter is observed, the line describes the general path of the flight (see Figure 10). If the tail rotor position is observed, stability of helicopter rotation can be visually assessed (see Figure 9).

Shadow. The shadow from the helicopter is a projection of its parts on the ground plane. It helps to determine exact position of the helicopter in the horizontal direction.

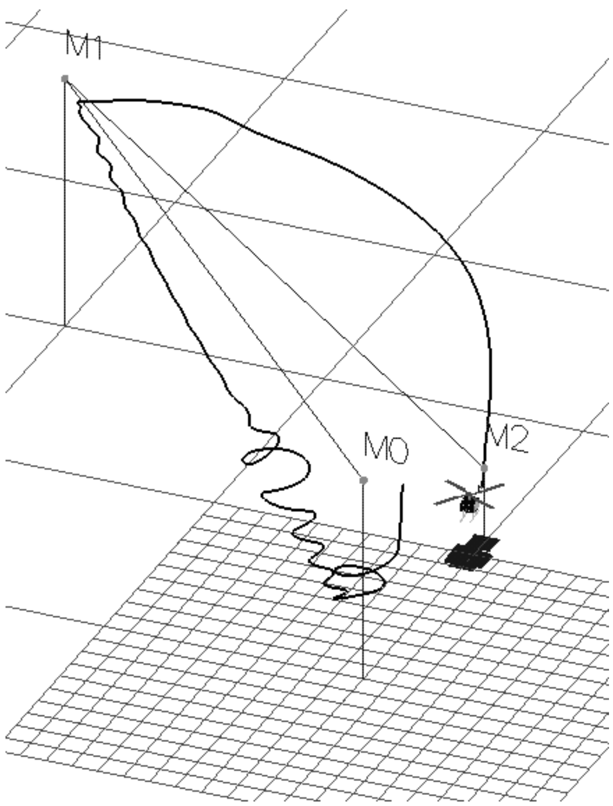

Figure 10: The whole trajectory compared to the mission path.

2.5 Conclusions on Helicopter Simulation and Visualization

The helicopter modeling, simulation and visualization project was the first large model obtained by the SolidWorks–to–Modelica compiler. It turned out to be possible to change the design of the helicopter in the CAD tool and rapidly produce a corresponding Modelica model.

The generated model can be conveniently connected to other, non-generated components, such as landing, ground model and load model. This added some more complexity to the set of equations to be solved.

The mechanical model of the helicopter was initially simulated separately from the control system. This is necessary in order to validate position of the center of mass with respect to the rotors, as well as to test functionality independent of the controller (e.g. landing). The model behaves as expected, and the solver computes the movements quite fast (10 seconds of flight is modeled using 5 seconds of CPU time[2])

When the model is connected to the external control system, various problems arise, both instability and performance problems. The trajectory of the flight, the success of mission, and the time it takes to perform the simulation depends on many aspects of the model and its simulation.

In many cases the mission was performed successfully, i.e. all the points were visited, the flight was stable, and the helicopter landed in a nor-

[2]Here and in further experiments we were using a 266 MHz Pentium II-based computer without graphic card, with 128 MB RAM.

mal vertical position at the landing point. Simulation time has been close to the real time (30-50 seconds of CPU time is necessary for simulation of 30 seconds of flight).

The simulation currently fails in the following three cases:

- The control system loses control over the horizontal position of the helicopter and cannot steer it to the target position. The helicopter flies away from the target.
- The control system loses control over the vertical or/and rotational position of the helicopter. It chaotically applies various forces and angles, but the helicopter falls down anyway.
- The adaptive solver takes too small integration steps so that the simulation takes 30 times more than the real time, i.e. 1 minute of flight is modeled using 30 minutes of CPU time.

The following factors affect the failures:

Control system simulation step size. The step size in the original model was 0.01 sec. Variations of this value (0.02, 0.03) sometimes caused failures, sometimes removed the failures.

Mechanical system solver step size. Larger steps typically causes less stability and faster simulation.

Mechanical system solver precision. Larger precision threshold caused less stability and faster simulation.

Mechanical coefficients. A number of unknown coefficients were guessed, such as density of material, speed of rotation of the rotors, weight of the rotors, etc. Small changes of these coefficients affect the failures.

Mission properties. Sharp corners of the mission line and short distances between the points caused failures. Actually all the failures occurred in the case of very short flights (less than 30 seconds), and typically due to small distance between mission points (less than 20 m). All experiments with larger missions were much more successful. However they were not so spectacular for visualization and not so convenient for validation. From this we conclude that the model has no enough time to react properly to the often changes in the prescribed flight direction.

As a result of the simulation some errors in the control system were identified. Some coefficients used in this system were adjusted. Therefore the goal of the experiments was partially achieved. However the mechanical model is certainly too simplified. Currently the interaction between the helicopter and the air is modeled by a single lifting force. More accurate models should be created including aerodynamic properties of rotor blades.

3 Robot Modeling

A study of service robots based on modeling and simulation was initiated in the framework of the WITAS [15] project. The major objective of the project was to predict tasks that a service robot-manipulator cooperating with unmanned autonomous aircraft will be able to solve in the future. The service tasks that the robot will perform in the future can, for instance be the following:

- Refueling the helicopter.
- Loading and unloading of cargo.
- Exchange of components of the helicopter.
- Minor repairs.

The initial plans contained a study of cooperation and interaction between a service robot and a helicopter. However in a typical situation the task of the robot can be described as ”to take some load from one container and to place it to another container”. A helicopter can serve as one of these containers. The position and rotation of the containers are known to the control system of the robot. Finally the robot modeling problem was formulated the following way:

- Input:
  - Given positions of containers (later also slots) and their identifiers, such as `c1` or `c2`.
  - Given a sequence Robot Command Language (RCL) commands (missions), such as `MOVE(c1,c2)`.
  - Given a virtual robot constructed in SolidWorks[14].
- Output:
  - Torques applied by the motors in order to perform the mission, for instance, to grab a load from container `c1` and move it to container `c2`.
  - The movement trajectory of the robot and all its joints when the mission is performed.

3.1 Mechanical Part of the Robot and the Load Model.

The mechanical part of the robot model was initially designed in SolidWorks and then automatically translated to Modelica. The model contains seven rigid bodies with six rotational joints between them. There is a motor controlled by the control system at each joint. In addition, the robot is placed on a mobile platform. The mobile platform is located on the ground (floor) and has two translational joints. All the joints are steered by motors, which are controlled by the controller. The platform can reach any point on the plane. From the platform the robot can reach any point in the volume $2 \times 2 \times 1\ m$ above the center of the platform.

The robot can interact with the load. Loads represent spare parts or cargo. The load is a cube of size $0.1\ m$ in each dimension. There are three connectors on the grip of the robot, which correspond to three connectors on the load. When the grabbing mode is turned on, three stiff springs-and-dampers are activated between the grip and the load. After some interaction the robot keeps the load firmly in its manipulator.

The load has a simplified model of collision detection with the ground. When the load reaches the ground, and some of the vertices of the load occur under the ground a stiff spring-and-damper is activated and pushes the vertex away from the ground. After some interaction the load rests still on the ground.

3.2 Environment Model

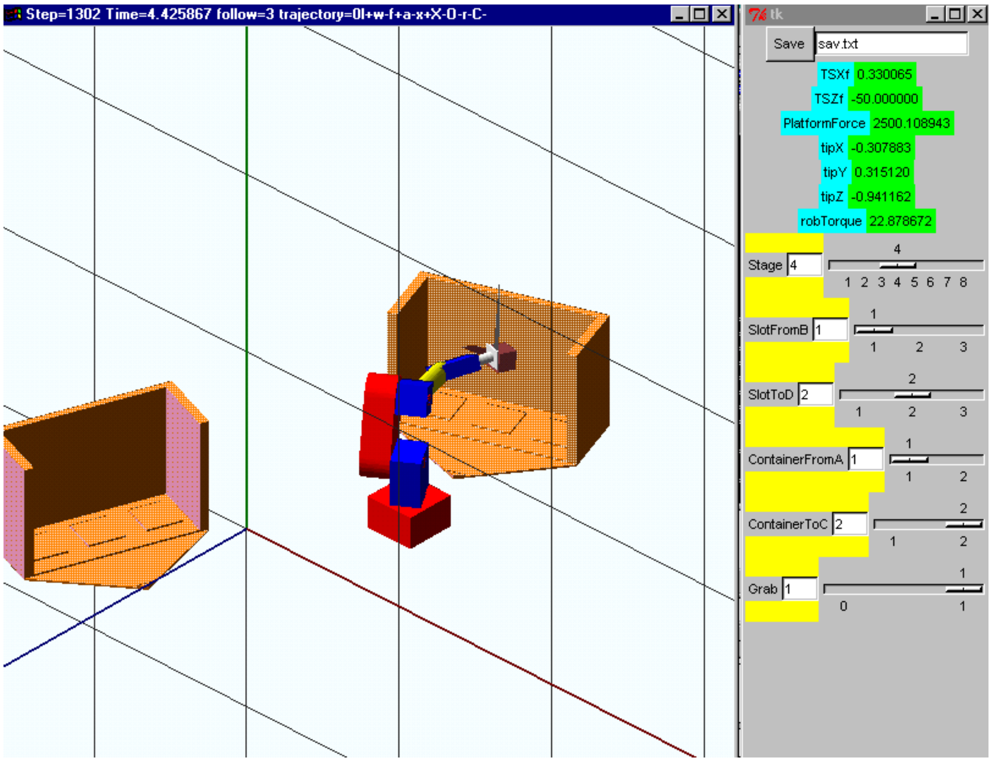

Figure 11: A virtual environment includes a robot, load (a small cube) and two containers.

The operating environment consists of two or more containers (see Figure 11). Containers represent helicopters, or storehouses with spare parts or cargo. Each container has an identifier and it contains three or more slots which have their identifiers. Containers are constructed as open boxes (approximately $0.5 \times 0.5 \times 0.5$ m) with the top and front wall missing. The slots have fixed positions within the containers. The load can be placed at the slots or removed from them. We assume that all the containers in the simulated world are standardized, i.e. the slots have fixed position. The position and rotation of container is known to the control system.

3.3 Scenario for Load Movement

Step	Platform	Manipulator	Grabbing	Terminate if
0	origin	home	off	immediately
1	source	home	off	$D_{platf} < \varepsilon$
2	source	source	off	$D_{manip} < \varepsilon$
3	source	source	on	$D_{grab} < \varepsilon$
4	target	home	on	$D_{platf} < \varepsilon$
5	target	target	on	$D_{manip} < \varepsilon$
6	target	target	off	immediately
7	origin	home	off	$D_{platf} < \varepsilon$

Table 1: Steps of scenario for moving a load from source to target.

The scenario for load movement is shown in the Table 1. The controller switches from one step to another when certain conditions become true. At each step specific target values for the actuators are specified. The platform motors can move on the plane between the origin position, source and target container. Every container has a reference point where the platform should be located. The actuators for the platform compute force necessary to move the platform in needed direction.

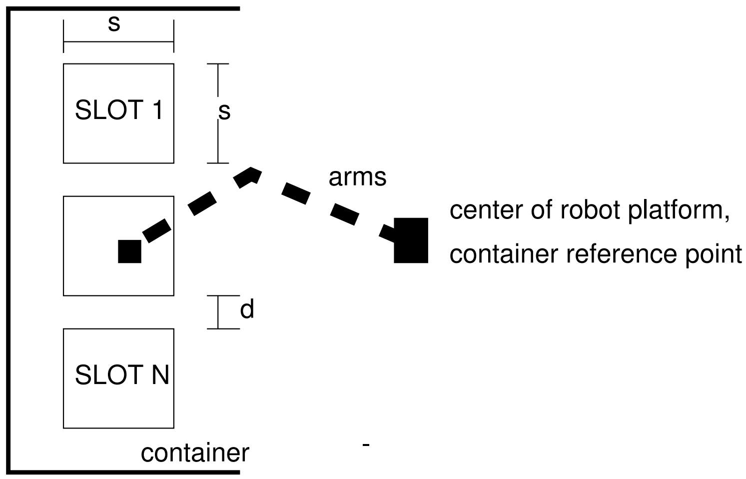

Figure 12: Container with three slots. View from above.

The manipulator can be in three positions: home, source slot and target slot position. In the home position the grip is at a position about 1 m above the ground. The slot positions are shown in Figure 12. The slot position for each slot (relative to the container reference point) is known from the container geometry. However since the robot has six revolute joints, the inverse geometry problem should be solved in order to move the grip to the necessary position and rotation. We discuss our solution to inverse geometry problem in Section 3.4.

The grab flag turns on and the stiff spring between the manipulator grip and the load, modeling a magnet-based manipulator.

The termination conditions are defined by functions D_{platf}, D_{manip} and D_{grab}. If the difference between required value and actual state (angles of the revolute joints, position of the prismatic joint, length of the spring) is close to zero, and its derivative is close to zero, we consider that the required value is reached. It appeared to be difficult to derive appropriate values for ε for each case. Therefore the termination condition is computed "visually" by the user, and he or she can interactively control the simulation by giving an order to shift to the next step of the plan.

The plan is implemented as an array in Modelica, and the value of variable for indexing the array is obtained via a graphical user interface (GUI) input function.

In order to get the new value from the GUI, a sampling method is used:

```
when sample (0, 0.1) then
 step=infun(...)
end when;
```

Every 0.1 seconds the integration stops, a new value from the GUI is obtained, and integration restarts with a possibly different target value for the new step according to Table 1. If the value from the GUI is still old, no variable change occur, and therefore the integration restart does not take much time from the solver.

3.4 The Inverse Geometry Problem

Objects in the simulated world can be easily located using three Cartesian coordinates, and their rotation is defined by additional three angles. When robot links are connected by revolute joints, the position and rotation of the grip of the robot is defined by six angles. The inverse geometry problem in our case is finding such angles of the robot joints, so that the grip is located at the required position (in Cartesian coordinates) and the required rotation.

3.4.1 The Inverse Robot in 2D

The 2D case is illustrated in Figures 13 and 14. In Figure 13 a manipulator (direct robot) is shown. It has two motors, two bars (links) and the tip. The position of the tip (X, Y) is defined by the angles A and B. Therefore we call such model a *direct* robot.

The inverse geometry problem is to determine values for A and B from the given X and Y. For this purpose the model depicted in the Figure 14 has been constructed. The angles (joints) A and B are free now. In addition a revolute joint appears at the tip. Two prismatic bars with actuators are added to the model. One bar moves along the X axis. Another bar is attached to the first one and moves along the Y axis. The positions of these

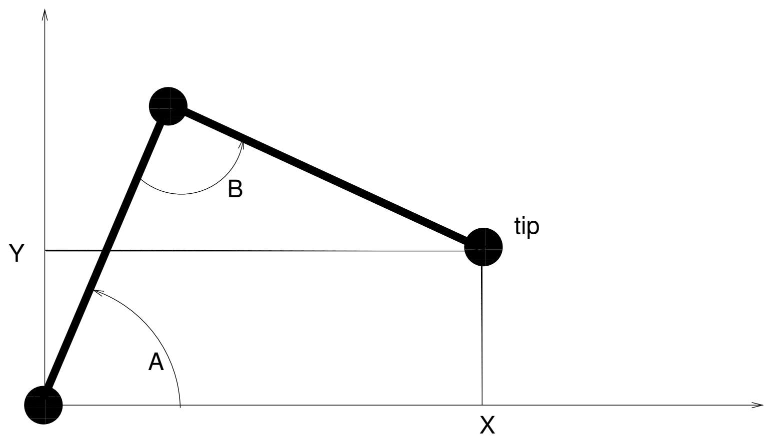

Figure 13: Direct kinematics model for two links with two revolute joints.

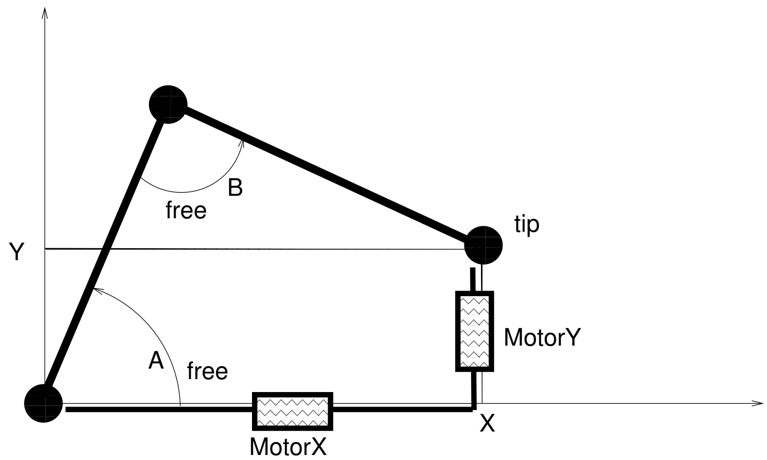

Figure 14: Inverse kinematics model for two links with two revolute joints.

bars define the (X, Y)–position of the robot tip. The angles at the revolute joints A and B can be obtained. This model is called an *inverse* robot.

3.4.2 Alternative Solutions

Generally, the inverse geometry problem can be solved by trigonometric computations in 3D, and requires complete information about the kinematics of all the links (i.e. the positions of the joints and their types). The inverse geometry problem is a special case of the inverse kinematics problem where movement of a point in Cartesian coordinates should be mapped to the change of angles in revolute joints. The inverse geometry problem is solved by finding a Jacobian, which describes the relations between the changes. It might appear that the solution is missing or that the process of solution reaches a singularity point.

3.4.3 The Inverse Robot in 3D.

In our approach we use a kinematic loop in order to obtain necessary values for the inverse geometry. For this purpose a corresponding model of the inverse robot is created from original (direct) robot model. This model has the same geometry as the original robot, but is controlled in a different way.

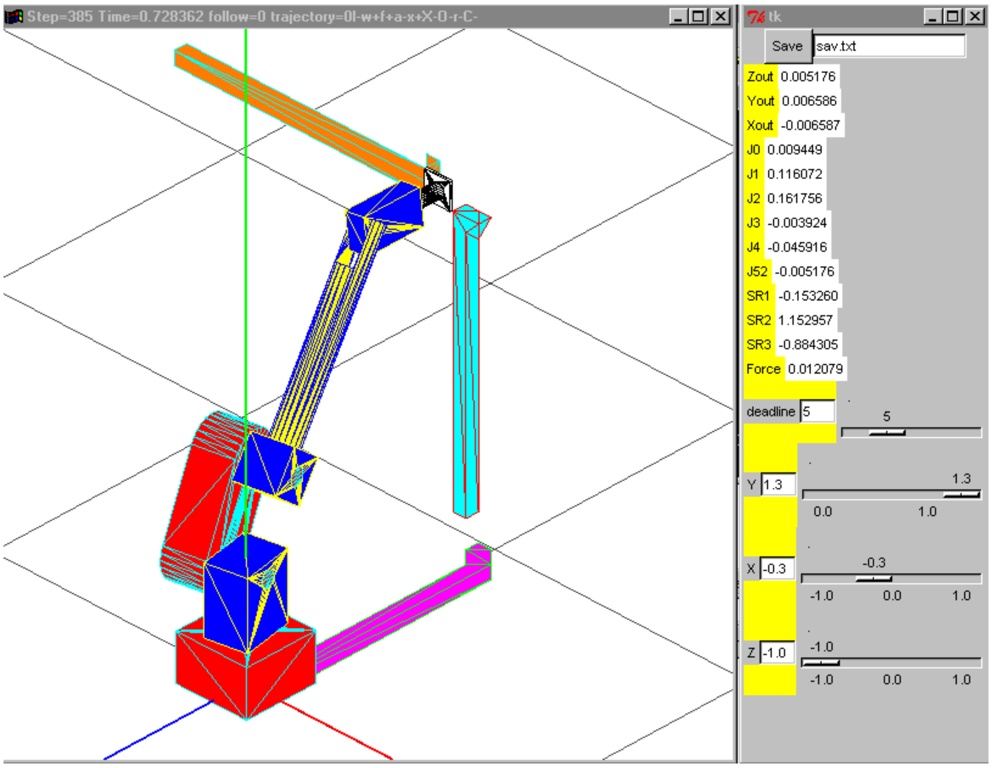

Figure 15: The inverse robot with additional bars has a kinematic loop. Required position of the tip is set up by the three scale bars (for X, Y, and Z).

The direct robot has six revolute joints with state variables. The inverse robot (Figure 15) has five revolute joints without state variables and one revolute joint `RevoluteCut3D`[3]. instead. These joints connect the links from the base to the grip. In addition three prismatic joints with state variables are added, and their movement is visualized by three bars. The reference position of the prismatic joints (scale bars for X, Y, and Z position) can be controlled interactively, just like the motors *MotorX* and *MotorY* in Figure 14.

The inverse geometry problem is solved in the following way. Initially all the joints stay in the home position. The user interactively specifies reference positions for the three orthogonal, consequently connected bars. Actuators apply forces to the bars causing these to move to the specified position. Since the inverse robot grip is attached to the last bar, all the joints

[3]The model `RevoluteCut3D` is a model of a revolute joint with one rotational degree of freedom. In the MBS library[2, 6] when kinematic loops occur such a joint model should be used instead of the usual revolute joint model `RevoluteS`.

of the robot move correspondingly. Finally the grip reaches the target and the angles $(\alpha_1, ..., \alpha_6)$ at all the revolute joints of the robot are stored for future use.

At this stage we do not need the inverse robot any more. We start simulation of the direct robot, which has motors attached to the revolute joints.

When the angles $\alpha_1, ..., \alpha_6$ are used as reference angles for the joints of the direct robot, the grip of this robot will gradually be moved to the required position. The same angles $\alpha_1, ..., \alpha_6$ can potentially be applied to the joints of a physical robot.

The inverse kinematics problem is solved just as a slight extension of the method described above. If the required position of the three prismatic joints is given as a function of time, the corresponding values of the angles for a certain time span can be saved, and used later in order to directly steer the direct robot.

3.5 Conclusions on Robot Simulation

A framework for mechanical simulation of manipulators has been developed. We considered a specific manipulator (with six rotational degrees of freedom, on a mobile platform), but the method can also be applied to other virtual devices. We show that a simplified deliberative and reactive layer of control for a robot can be designed using Modelica. Alternatively to table-driven plans, the robot operation plan can also be composed of a set of connected primitive plan steps.

Currently the plan is hard-coded in the Modelica model. As an alternative, it can be obtained from the interactive environment during simulation, or read from an input file. In this way programmable virtual robots can be modeled.

If inverse and a direct robot are integrated into the same simulation it is possible to give robot commands which include Cartesian coordinates. This greatly simplifies construction of the movement plan.

The disadvantages with our approach are that it can under certain circumstances be rather slow, and can, just as other methods, reach a point of singularity. Also, it is difficult to handle cases when the required position is out of reachable space for the robot.

The advantage is that the same model (or a model with small and well-defined modifications) can be used both for the direct and the inverse kinematics.

4 Acknowledgments

The Modelica definition has been developed by the Eurosim Technical Committee 1 (Modelica Design Group)[7] under the leadership of Hilding Elmqvist (Dynasim AB, Lund, Sweden) and Martin Otter (DLR, Germany). The Dymola tool has been designed by Dynasim AB. The Multibody Simulation Library has been developed by Martin Otter.

This work has been supported by the Wallenberg foundation as part of the WITAS project [15].

Johan Parmar[8] developed early variants of helicopter models in Modelica and integrated them with the controller written by Erik Skarman [13]. Tommy Persson ported this controller from Ada to C++.

References

[1] 3D Systems, *Stereo Lithography Interface Specification*, 3D Systems, Inc., Valencia, CA 91355. Available via http://www.vr.clemson.edu/credo/rp.html.

[2] Hilding Elmqvist, Dag Brück, Martin Otter, *Dymola, Dynamic Modeling Laboratory, User's Manual, Version 4.0*, from Dynasim AB, Research Park Ideon, Lund, Sweden, http://www.dynasim.se

[3] Vadim Engelson, *Integration of Collision Detection with the Multibody System Library in Modelica.* Linköping Electronic Articles in Computer and Information Science, ISSN 1401-9841, Vol. 5 (2000): nr 010. Available at: http://www.ep.liu.se/ea/cis/2000/010/

[4] Vadim Engelson, *Tools for Design, Interactive Simulation, and Visualization for Dynamic Analysis of Mechanical Models*. Linköping Electronic Articles in Computer and Information Science, ISSN 1401-9841, Vol. 5 (2000): nr 007. Available at: http://www.ep.liu.se/ea/cis/2000/007/

[5] Håkan Larsson, *Translation of 3D CAD Models to Modelica*, Master Thesis, LiTH-IDA-Ex-99/30, IDA, Linköping Univ., Sweden, March 1999.

[6] Martin Otter, Hilding Elmqvist and François E. Cellier, Modeling of Multibody Systems with the Object-Oriented Modeling Language Dymola, *Nonlinear Dynamics*, 9:91-112, 1996, Kluwer Academic Publishers.

[7] Modelica Design Group, *Modelica WWW site*, http://www.modelica.org

[8] Johan Parmar, *Modeling an Autonomous Helicopter and its Maintenance Environment Using Modelica*, Master Thesis, LITH-IDA-Ex-99/63, IDA, Linköping Univ., Sweden.

[9] PELAB Group, *Modelica activities in PELAB*, Linköping University, http://www.ida.liu.se/˜pelab/modelica

[10] L.R. Petzold, A description of DASSL: A differential/algebraic system solver, in *Proceedings of 10th IMACS World Congress, Montreal, August 8-13, 1982*.

[11] Erik Skarman. *An Aircraft Model*, Linköping Electronic Articles in Computer and Information Science, ISSN 1401-9841, Vol. 4 (1999): nr 013. Available at: http://www.ep.liu.se/ea/cis/1999/013/

[12] Erik Skarman. *An Helicopter Model*, Linköping Electronic Articles in Computer and Information Science, ISSN 1401-9841, Vol. 4 (1999): nr 014. Available at: http://www.ep.liu.se/ea/cis/1999/014/

[13] Erik Skarman. *A Helicopter Control System*, Linköping Electronic Articles in Computer and Information Science, ISSN 1401-9841, Vol. 4 (1999): nr 015. Available at: http://www.ep.liu.se/ea/cis/1999/015/

[14] SolidWorks Corporation, *SolidWorks*, http://www.solidworks.com

[15] WITAS group, *The WITAS home page*, http://www.ida.liu.se/ext/witas