IAA-AAS-DyCoSS2 -14-14-07

EMBEDDED MPC FOR SPACE APPLICATIONS

Carlo A. Pascucci[*], Alberto Bemporad[†], Samir Bennani[‡], and Max Rotunno[§]

Autonomy is being defined as the capability of a vehicle by means of its on board systems to perform functions without external support. Focusing on stabilization and guidance, in this paper we investigate the use of the Model Predictive Control (MPC) technique as a candidate technology to help bringing more autonomy to future space systems. By means of a multi-rotor Unmanned Aerial Vehicle (UAV) case study we detail the software and the hardware aspects concerning a fast real time MPC implementation for low level GNC functions on a low power embedded computing platform. Our control scheme performances are assessed through simulations, on-board testing and comparisons with other techniques.

INTRODUCTION

To enable significant advances in space exploration, mission requirements are getting increasingly challenging and harder to satisfy, moreover long distances limit communication with the ground station, thus a growing need for *autonomy* in space systems. This specific requirement can be achieved taking advantage of optimal control strategies like Model Predictive Control (MPC). A survey on MPC for aerospace applications can be found in references from[1] to[11] . From the software side, MPC is a systematic design approach for controlling multivariable systems. It is based upon the minimisation of a quadratic cost function and it is aimed to maximise the performance of the system while taking in account for its operational constraints. A Quadratic Program (QP) solver is the core of this control framework and its performance in terms of speed and accuracy deeply affects the quality of achievable autonomy. Up to date solvers complexity, coupled with poor computational capabilities in low power microcontrollers, prevented the adoption of MPC in the aerospace industry for low level GNC functions like stabilization, which is subject to faster dynamics than the outer loop for the trajectory planning and tracking task. However recent mobile computing platforms such as ARM Cortex processors and new solvers like Dual Accelerated Gradient Projection (GPAD)[12] can invert this trend bringing MPC capabilities also to complex and fast aerospace systems. The aim of this work is to demonstrate the feasibility of a fast real-time MPC implementation using an UAV low level stabilization and guidance problem as a case study.

In the following sections, after a brief review of the UAV system used and of its dynamics, the MPC problem formulation will be discussed along with comparisons among different QP solvers. After assessing through simulations the performance of our control framework, we will go through the embedded software development and testing on the selected hardware platform. Finally some considerations about the ongoing work and the future steps to be taken will be drown.

[*]Research Collaborator, DYSCO, IMT Lucca, Piazza San Ponziano 6, 55100 Lucca Italy.
[†]Full Professor, DYSCO, IMT Lucca, Piazza San Ponziano 6, 55100 Lucca Italy.
[‡]GNC System Engineer, ESTEC, ESA, Keplerlaan 1 Noordwijk The Netherlands.
[§]A3R,Via Enrico Ortolani 102 00125 Rome Italy .

UAV SYSTEM AND DYNAMICS

To assess the performance of MPC for future autonomous GNC space applications we have started with a simplified control benchmark. We designed a controller for stabilization and guidance for a Sixton UAV system provided by A3R [*]. It is a rotating wing class aircraft with six propellers arranged in three coaxial counter-rotating couples as shown in Figure 1.

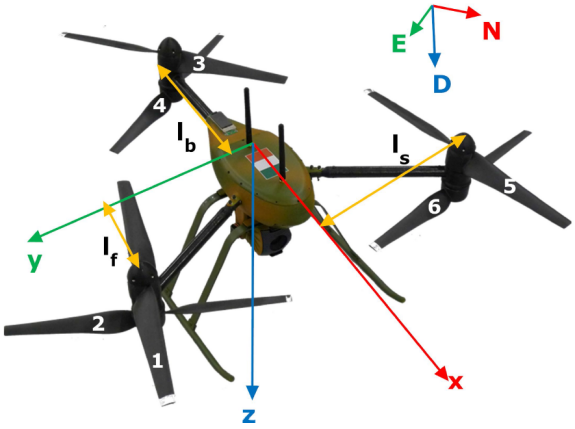

Figure 1. The Sixton UAV provided by A3R

This is an underactuated system defined by a highly coupled and nonlinear dynamics. Moreover it has both constraints on the state and on the inputs. Hard constraints are present on the input, represented by maximum and minimum motors rotational speed. Upper and lower propellers have different dimensions, this leads to two different set of constraints as stated by Equation (1)

$$1400 \leq u_{1,3,5} \leq 2700 \tag{1a}$$

$$1500 \leq u_{2,4,6} \leq 2900 \tag{1b}$$

where the motors rotational speed $u_i, i = 1, \ldots, 6$ are expressed in Rotations Per Minute $[RPM]$. Soft constraints instead are summarized in Equation (2). Them act on the state and set limits for roll (ϕ) and pitch (θ) angles expressed in radians $[rad]$, but also on minimum altitude which is lower bounded by the ground:

$$-\frac{\pi}{4} \leq \phi \leq \frac{\pi}{4} \tag{2a}$$

$$-\frac{\pi}{4} \leq \theta \leq \frac{\pi}{4} \tag{2b}$$

$$h \geq 0 \tag{2c}$$

The aim is indeed to control attitude and altitude through a Linear Time Invariant (LTI) Model Predictive Controller, hence the derivation of the implicit MPC solution is based on the model dynamics linearised around the equilibrium point represented by the hovering condition as shown by Equation (3)

[*] http://www.a3r.it/

$$\frac{J_x}{\ell_s}\ddot{\phi} = T_u(\tilde{u}_5 - \tilde{u}_1) + T_d(\tilde{u}_6 - \tilde{u}_2) \tag{3a}$$

$$\frac{J_y}{\ell_f}\ddot{\theta} = T_u(\tilde{u}_1 + \tilde{u}_5 - \alpha\tilde{u}_3) + T_d(\tilde{u}_2 + \tilde{u}_6 - \alpha\tilde{u}_4) \tag{3b}$$

$$J_z\ddot{\psi} = Q_u(\tilde{u}_1 + \tilde{u}_3 + \tilde{u}_5) - Q_d(\tilde{u}_2 + \tilde{u}_4 + \tilde{u}_6) \tag{3c}$$

$$m\ddot{h} = T_u(\tilde{u}_1 + \tilde{u}_3 + \tilde{u}_5) + T_d(\tilde{u}_2 + \tilde{u}_4 + \tilde{u}_6) - mg \tag{3d}$$

where ϕ, θ, and ψ represent respectively the roll, the pitch, and the yaw angles, while h represents the altitude; J_x, J_y, and J_z are the diagonal elements of the inertia tensor; m is the mass of the vehicle, while g is the gravity; T_u and T_d are coefficients that relates respectively the upper and the lower propellers thrust to their angular velocity, and similarly Q_u and Q_d relates respectively the upper and the lower propellers torque to their angular velocity; α is the ratio between the characteristic dimensions of the Sixton UAV ℓ_b and ℓ_f; finally $\tilde{u}_i$ represents the i-th propeller's rotation value. All attitude angles are represented as radians $[rad]$, angular velocities as radians over second $[rad/s]$; the altitude value is intended to be in meters $[m]$, thus its derivative can be expressed as meters per second $[m/s]$; finally the control inputs units are Rotations Per Second to the power of two $[(RPS)^2]$.

LINEAR MODEL PREDICTIVE CONTROL

To formulate the linear MPC problem we used the same paradigm as in the MPCSofT toolbox,[13] A MATLAB/Simulink based software tool developed by IMT within previous ESA contracts. The control action at time k is obtained through Equations (4) and (5)

$$\begin{cases} x(k+1) = Ax(k) + Bu(k) + f \\ z(k) = E_z x(k) + H_z u(k) + P_z \Delta u(k) \\ c(k) = E_c x(k) + H_c u(k) + P_c \Delta u(k) \end{cases} \tag{4}$$

where $\Delta u(k) = u(k) - u(k-1)$ is the input increment, z is the "performance vector" to be optimized, c is the "constrained vector" and E, H, P are matrices defining the structure of the problem. The MPC optimal control problem solved at each sampling instant is

$$\begin{cases} \min \quad \rho_1\epsilon_1^2 + \rho_2\epsilon_2^2 + \sum_{k=0}^{N-1}(z(k) - r(k))'(z(k) - r(k)) \\ \\ subject\ to \\ \\ \Delta u(k) = 0,\ \forall k = N_u, ..., N \\ c(k) \leq c_{max} + V_c\epsilon_1,\ k = 0, ..., N-1 \\ C(N)x(N) \leq d(N) + V(N)\epsilon_2 \end{cases} \tag{5}$$

where $r(k)$ is the reference signal for vector $z(k)$, N is the prediction horizon, N_u is the control horizon, c_{max} is the vector including state and control inputs constraint values, V_c is a vector used to specify which constraint is soft and which one is hard, $C(N)$, $d(N)$ and $V(N)$ have the same meaning of c, $cmax$ and V_c respectively, but for the terminal stage, ϵ_1 and ϵ_2 are slack variables

used to soften the constraints, and ρ_1, ρ_2 are constant weighting terms. Equations (4) and (5) are then mapped into a Quadratic Programming (QP) problem described by Equation (6).

$$\begin{cases} \min \quad \frac{1}{2} w^\top H w + c^\top w + d \\ subject\ to \\ Gw \leq b \end{cases} \tag{6}$$

where w is the optimization variable, H is the Hessian (positive semidefinite and symmetric) matrix, c the linear weighting vector, d is a constant term, G is the matrix of linear constraints, and b is the vector of upper bounds on Gw.

The GPAD Quadratic Program Solver

To solve the QP problem we implemented in MATLAB the GPAD algorithm. The solver code is very compact, the core loop requires less than 20 lines, and it's easy to embed. Moreover it has advantages compared to other fast gradient methods such as the ability to handle arbitrary polyhedral constraints on inputs/states, polyhedral terminal sets, and to use practical termination criteria based on the primal solution. In addition, compared to Interior Point (IP) methods,[14] it involves only matrix vector products at each iteration instead of solving a linear system, can reach a solution of moderate accuracy ($\sim 10^{-3}$) quite fast (faster than a simple gradient method), and has much tighter theoretical complexity bounds then IP. It is worth noting that although the GPAD solver was not conceived for speed, it is still reasonably fast, and it shows the best scalability with the number of QP variables as can be assessed by the synthetic benchmark results depicted in Figure 2. The comparison has been made against the 64bit *mex* version of three commonly used solvers such as the MathWorks' QUADPROG, DANTZGMP, and QPKWIK included in the Optimization Toolbox, and the MPC Toolbox v3 and v4 respectively. The machine used was equipped with an Intel Core i7 920 processor. The following code snippet was used to create the QP problems in the benchmark:

```
H=rand(n,n); H=H*H;H=(H+H)/2+.1*eye(n);
c=10*rand(n,1);
G=[eye(n)+rand(n,n);-eye(n)-rand(n,n)];
b=[rand(n,1);rand(n,1)]+0.1;
```

To achieve precise and reliable time measurements, due to the multi-core and multithreaded architecture, the Matlab's *tic*, *toc* paradigm has been used. For each n 20 random QPs are generated, each one has been solved 10 times, and the timing results averaged to minimize the effect of measurements variance due to the non real-time operating system.

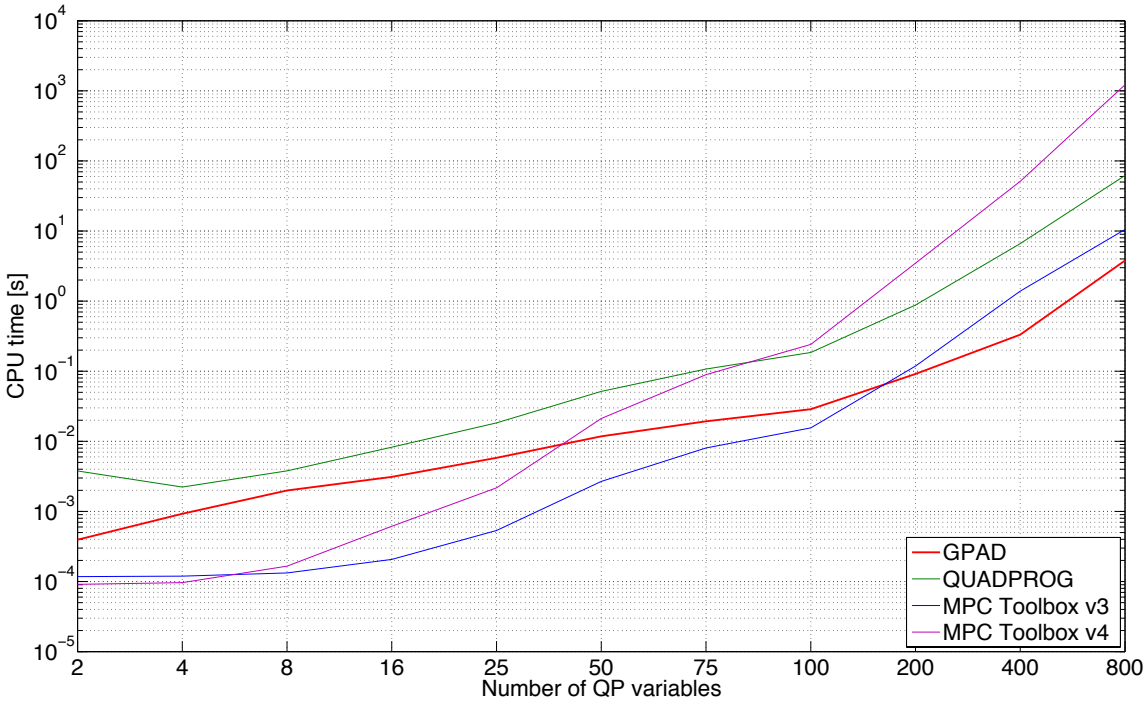

Figure 2. Graphical comparison of QP solvers

SIMULATION RESULTS

While the derivation of the implicit MPC control solution is based on the linearised model dynamics, all validation tests and simulations are performed taking in account for the nonlinear model of the aerial vehicle. To assess our controller's performance we executed three reference tracking tests. The first is the nominal case, thus without any external disturbance, in the second case we added an upward pointing wind gust, and in the third one we simulated a power battery voltage drop to test our solution robustness to unmodeled events. The setpoint to track is reported in Equation (7).

$$\left\{ h = 5\,[m] \quad , \quad \phi = \frac{\pi}{12}\,[rad] \quad , \quad \theta = -\frac{\pi}{12}\,[rad] \quad , \quad \psi = \frac{\pi}{4}\,[rad] \right. \tag{7}$$

Relevant MPC parameters are: the prediction horizon $N = 20$; the control horizon $N_u = 1$, and the sampling time $Ts = 0.01[s]$; while the state weighting in the cost function is $W_\phi = 10, W_{\dot{\phi}} = .1, W_\theta = 10, W_{\dot{\theta}} = .1, W_\psi = 20, W_{\dot{\psi}} = 1, W_h = 10, W_{\dot{h}} = .5$.

In Figure 3 simulation results for the nominal case are shown, where ϕ, θ, and ψ are the roll, the pitch, and the yaw angle expressed in radians respectively, and h is the altitude expressed in meters, $MV_i, i = 1, \ldots, 6$ represents the controller's input to the system. To better compare our work with the existing technology, we used the Sixton's A3R default controller as baseline, whose behavior from now on is depicted on the right side of the Figures. Dashed lines represents the desired state values, while the solid ones stand for the actual state value. We chose the reference detailed in Equation (7) because it requires the controller to satisfy multiple competing objectives concurrently. Climbing from $h = 0[m]$ to $h = 5[m]$ requires to speedup all the motors evenly, to

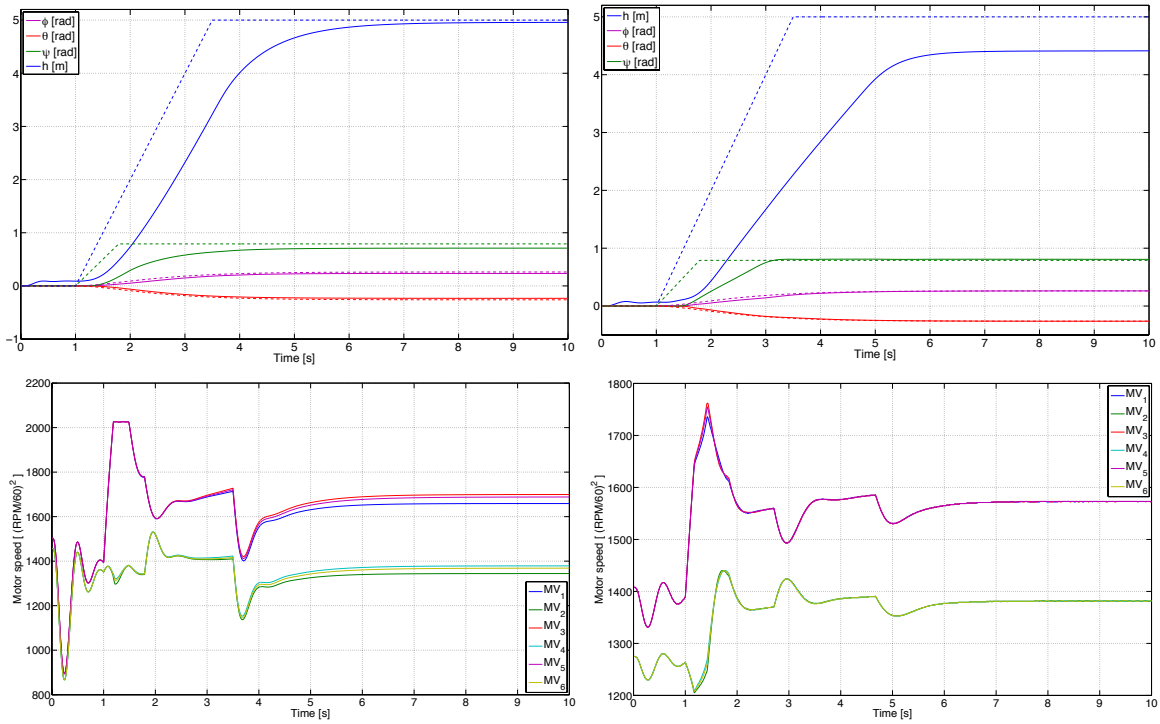

Figure 3. Nominal case: MPC (left side) vs A3R's baseline controller (right side)

pitch, roll, and yaw instead, different momenta and torques have to be generated at the same time. It is worth noting that the initial h offset is due to altitude sensor positioning, which is placed on top of the landing gear at $0.10[m]$ above the ground.

We then tested the capability of the controller from A3R to hold the altitude in presence of wind. Since in this implementation there is no position holding on x and y axes we simulated a wind gust acting on the z axis. Starting to blow at time $T = 10[s]$, the wind will reach the maximum speed of $5[m/s]$ at time $T = 20[s]$. When the blowing direction is upward, like a thermal wind, the controller slows down the propellers rotational speed to try keeping the altitude. Figure 4 shows how the MPC controller reacts to the unmodeled disturbance while tracking the desired state reference.

Figure 5 shows the results in case of a simulated battery discharge causing the voltage to drop from the nominal value of $14[v]$ to $11.9[v]$. This affects directly the maximum motors rotational speed, thus, to generate the required forces, while actual propellers RPM does not have to change, Manipulated Variables (MV) values should increase to compensate the power loss. When hovering with a battery voltage of $14[v]$ the commanded motor input is $u_{1,3,5} = 1546[(\frac{RPM}{60})^2], \quad u_{2,4,6} = 1362[(\frac{RPM}{60})^2]$, but when the battery voltage drops to $11.9[v]$ the commanded motor input become $u_{1,3,5} = 1651[(\frac{RPM}{60})^2], \quad u_{2,4,6} = 1437[(\frac{RPM}{60})^2]$, while the actual propeller speed remain always $u_{1,3,5} = 2218[RPM], \quad u_{2,4,6} = 2183[RPM]$.

It is worth noting that in all test cases, expecially in the second and in the third one, commanding a more aggressive reference tracking (e.g. requiring greather pitching and rolling angles), causes the baseline controller to fail, differently from the MPC that can exploit effectively all the workspace defined by the UAV's constraints.

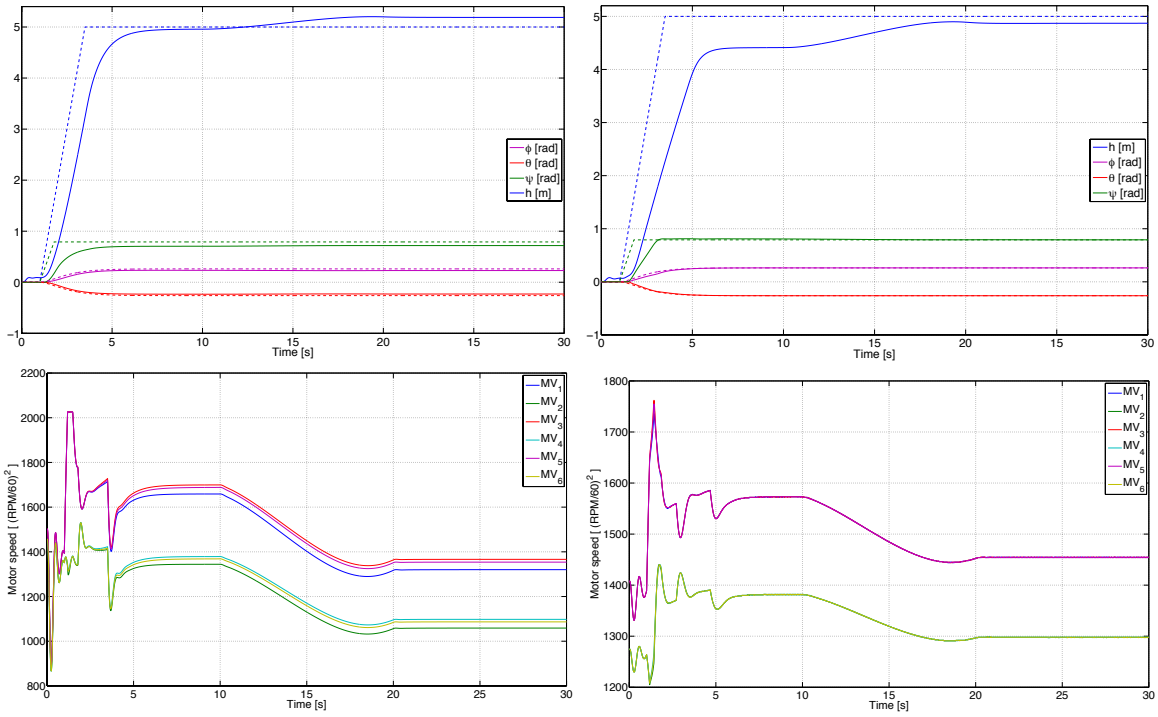

Figure 4. Upward pointing wind: MPC (left side) vs A3R's baseline controller (right side)

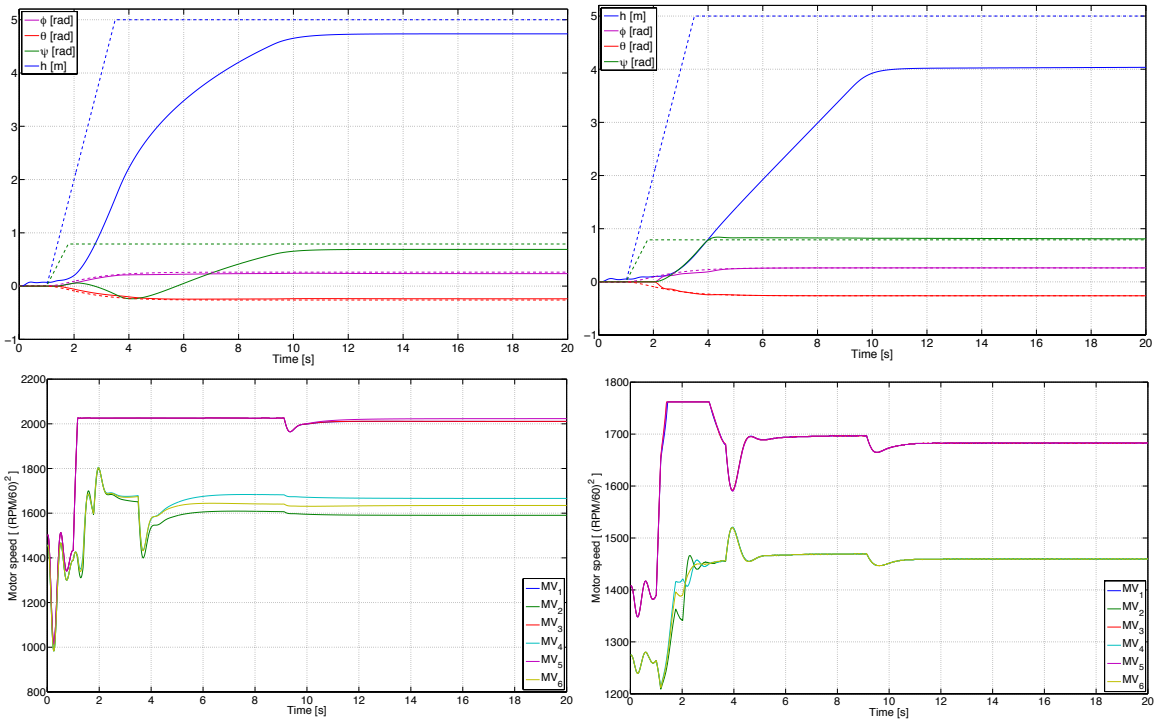

Figure 5. Battery voltage drop: MPC (left side) vs A3R's baseline controller (right side)

HARDWARE IMPLEMENTATION

To use our MPC control scheme in a a real world scenario, taking advantage of the Embedded MATLAB (EML) language flexibility, by means of the Simulink's Automatic Code Generation procedure, we obtained embeddable C code for the MPC controller using the GPAD solver. The selected embedded platform has been the Beagleboard-xM by Circuitco. The board is based upon the Texas Instruments DM3730 SoC. It includes in the same package a 1-Ghz ARM Cortex-A8 CPU, a POWER SGX graphics accelerator, a C64x DSP and 512MB LPDDR RAM. Being $100Hz$ the sampling frequency of our system, the hard real-time constraint consists in solving the MPC problem in less than $10[ms]$. The generated code has been optimized for the ARM Cortex CPU. The installed operating system was Ubuntu 13.04 core, compiled to exploit floating point numbers on hardware. The execution time has been measured directly on the Beagleboard-xM, calling the `clock_gettime()` function, inside the *main* function, with the `CLOCK_MONOTONIC_RAW` option that provides access to the raw hardware-based time, that is not subject to adjustments performed by other functions. Besides how the code is written, a crucial aspect to consider is the compiler. Although commercial closed source ARM optimized compilers are available, we used *gcc* (v 4.7.3) as it is known to be a *de facto* standard for the Linux environment. Depending on the optimization flags the code execution speed can vary significantly, thus for the sake of clarity we report here the used compiler's directives:

```
CFLAGS = -Wall -O3 -march=armv7-a -mfloat-abi=hard -mfpu=neon
         -funsafe-math-optimizations -ftree-vectorize
         -ffast-math -funroll-loops -lrt -I$(IDIR)
```

To assess code's performance we replicated on-board the setpoint tracking test described in the previous section. In the worst case the QP problem requires to the GPAD algorithm 340 iterations to converge to a solution. Without optimization flags the CPU time is $378.40[ms]$. It is worth noting that the same code in the worst case takes $2.91[ms]$ of CPU time in the test machine used for the simulations in the previous section. Activating compiler's flags in the Beagleboard-xM reduces the time to $92.82[ms]$ which is a considerable speedup, but it is not enough to our purposes. It is worth noting that up to now all the calculations have been done in double precision. This level of numeric accuracy can be cumbersome for this kind of embedded microprocessor. Although an hardware floating point unit is available, it supports only single precision variables indeed. Switching from double to single precision will almost double the board's performance, reducing the measured execution times that now is $47.76[ms]$. In spite of this the code is still running almost five time slower than we need to satisfy the $10[ms]$ real-time constraint. This is due to the QP's dimension. This depends on the prediction and control horizons (N, N_u), the number of system's states and control inputs (nx, nu), and the number of constraints, including terminal ones (nc, ncN), plus two additional items related to the presence of soft constraints. More formally

$$H \in R^{nvar \times nvar} \quad c \in R^{nvar} \quad G \in R^{ncon \times nvar} \quad b \in R^{ncon} \tag{8}$$

where $nvar = nu * N_u + 2$ is the number of optimization variables in the QP, and $ncon = N * nc + ncN + 2$ is the number of constraints in the QP. Referring to Equation (6), all the constraints have to be defined in the $Gw \leq b$ format, hence are present twelve hard constraints arising from the motors, plus five soft ones due to the state variables, leading to seventeen constraints globally.

In our simulation setup $H \in R^{8\times8}$, $c \in R^8$, $G \in R^{343\times8}$, and $b \in R^{343}$, thus $M \in R^{343\times343}$ is the dimension of the dual's Hessian. Being the GPAD a dual method, the dimension of M deeply affects the computational speed. Without affecting the prediction horizon N, we can reduce $nvar$ by introducing a constraint's prediction horizon N_{cy}, then considering a new $ncon = N_{cy} * nc + ncN + 2$. Setting $N_{cy} = 5$, leads to $G \in R^{87\times8}$, and $b \in R^{87}$, thus $M \in R^{87\times87}$. This is a considerably smaller problem that takes $11.76[ms]$ with 340 iterations. To further reduce the computational time, another aspect we can take in account is the number of iteration required to converge to a solution. This variable depends on the cost function value. As stated in Equation 5, it depends on the distance between the actual and the desired setpoint at each time step. A meaningful reference should be related at the UAV system's physical specifications. Due to the total motors thrust, the Sixton's maximum climb speed is $10[m/s]$. Similar considerations can be done also for the angular rates. Hence smoothing the user references through appropriate slew rate functions reduces significantly the number of required iterations while not affecting the quickness in the UAV response. Applying this technique for this QP problem we can make the GPAD to converge in only 42 iterations. This translates in $1.76[ms]$ of CPU time on the Beagleboard-xM, thus effectively satisfying our real-time constraint.

CONCLUSIONS

Our work assessed the feasibility of an MPC controller implementation for aerospace applications on a representative embedded platform. Although the worst case $1.76[ms]$ CPU time measured on the ARM Cortex-A8 is more than enough to satisfy the $10[ms]$ real-time constraint for the tested UAV, at the time of writing we are working also on a custom embedded MPC C code to fully exploit the ARM Cortex architecture capabilities in order to proceed with the extension towards autonomy. This could also allow to address bigger QP problems arising from more complex systems and greater constraints sets. It is worth nothing that, thanks to the flexibility of the MPC framework, the control action could be immediately extended to fully regulate the Sixton's spatial displacement by simply adding the needed equations in the linear model used to derive the controller. Moreover simply changing the weights in the cost function of the optimization problem, the user can choose to how control the vehicle's movements, by means of velocities rather than absolute angles and positions or even a mix of them. Being this work focused on a fast real-time MPC implementation, no frequency domain analysis to test robustness indicators such as gain, phase and delay margins is reported here, anyway we are going to perform it reformulating the controller as a set of explicit MPCs.

ACKNOWLEDGMENTS

The study is conducted under ESA NPI scheme "Model Predictive Control for Adaptable Space Applications (MPC4ASA)", Grant number 4000106153

REFERENCES

[1] A. Bemporad, C. A. Pascucci, and C. Rocchi, "Hierarchical and hybrid model predictive control of quadcopter air vehicles," *Analysis and Design of Hybrid Systems*, Vol. 3, 2009, pp. 14–19.

[2] A. Bemporad and C. Rocchi, "Decentralized hybrid model predictive control of a formation of unmanned aerial vehicles," *Proc. 18th IFAC World Congress, Milano, Italy*, 2011, pp. 11900–11906.

[3] G. Binet, R. Krenn, and A. Bemporad, "Model predictive control applications for planetary rovers," *Proc. 11th International Symposium on Artificial Intelligence and Robotics in Space (iSAIRAS), Sept*, 2012, pp. 4–6.

[4] M. Saponara, V. Barrena, A. Bemporad, E. Hartley, J. Maciejowski, A. Richards, A. Tramutola, and P. Trodden, "Model predictive control application to spacecraft rendezvous in Mars Sample & Return

scenario," *Proc. 4th European Conference for Aerospace Sciences (EUCASS), Saint Petersburg, Russia*, 2011.

[5] W. B. Dunbar, M. B. Milam, R. Franz, and R. M. Murray, "Model predictive control of a thrust-vectored flight control experiment," *15th IFAC World Congress on Automatic Control*, 2002.

[6] A. Richards and J. How, "Performance evaluation of rendezvous using model predictive control," *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, 2003.

[7] A. Richards and J. How, "Decentralized model predictive control of cooperating UAVs," *Decision and Control, 2004. CDC. 43rd IEEE Conference on*, Vol. 4, IEEE, 2004, pp. 4286–4291.

[8] E. Hartley, *Model predictive control for spacecraft rendezvous*. PhD thesis, Ph. D. Dissertation, University of Cambridge, UK, 2010.

[9] A. Alessio and A. Bemporad, "A survey on explicit model predictive control," *Nonlinear model predictive control*, pp. 345–369, Springer, 2009.

[10] M. Wood and W.-H. Chen, "Regulation of magnetically actuated satellites using model predictive control with disturbance modelling," *Networking, Sensing and Control, 2008. ICNSC 2008. IEEE International Conference on*, IEEE, 2008, pp. 692–697.

[11] Ø. Hegrenæs, J. T. Gravdahl, and P. Tøndel, "Spacecraft attitude control using explicit model predictive control," *Automatica*, Vol. 41, No. 12, 2005, pp. 2107–2114.

[12] P. Patrinos and A. Bemporad, "An accelerated dual gradient-projection algorithm for embedded linear model predictive control," 2012.

[13] R. Krenn, A. Gibbesch, G. Binet, and A. Bemporad, "Model Predictive Traction and Steering Control of Planetary Rovers," *ASTRA*, Vol. 1, No. 16, 2013.

[14] Y. Wang and S. Boyd, "Fast model predictive control using online optimization," *Control Systems Technology, IEEE Transactions on*, Vol. 18, No. 2, 2010, pp. 267–278.