Available online at www.sciencedirect.com

ScienceDirect

IFAC PapersOnline CONFERENCE PAPER ARCHIVE

IFAC PapersOnLine 55-1 (2022) 561–566

Optimization Techniques for Online MPC in Android Smartphones for Artificial Pancreas: A Comparison Study

Abishek Chandrasekhar and **Radhakant Padhi**

Dept. of Cyber-Physical Systems, Indian Institute of Science, Bangalore, India.

Abstract: Model Predictive Control (MPC) has evolved as a potential approach for safety-critical Artificial Pancreas (AP) systems because of its constraint handling capability. However, it demands a solution of a fairly high-dimensional constrained optimization problem online. A popular approach to close the loop in practice is to implement it in smartphones in general and in Android smartphones in particular (because of its pervasiveness). As MPC requires the use of Optimization solvers in smartphones, a survey of various available options has been done in this paper, followed by a thorough comparison study about their performance. This comprehensive comparison includes the ease of convergence of the optimization solution and the computational time. It turns out that the Python-based optimization solver SciPy is most suitable for this purpose as compared to other alternatives such as Casadi, lpsolvers and autocode generation process of MATLAB. The outcome of this study can be used for rapid prototyping of Android smartphone-based AP systems.

Keywords: Artificial Pancreas (AP), Embedded Control, Optimization, Model Predictive Control (MPC), Online Optimization, Smartphone Application

1. INTRODUCTION

Type-1 Diabetic Mellitus (T1DM) is a metabolic condition where the patient's insulin-producing beta-cells are destroyed, because of which, the patient is unable to regulate their blood glucose (BG) within normal ranges (between 70 mg/dL and 180 mg/dL). If the BG levels are below 70 mg/dL, it is termed *hypoglycemia* and if it exceeds 180 mg/dL, it is termed *hyperglycemia*. Hypoglycemia can prove to be quite dangerous as it can lead to the patient becoming unconscious, induce a coma, in some cases, it can even lead to death. Prolonged episodes of hyperglycemia cause significant health complications such as nephropathy, neuropathy, diabetic retinopathy, and cardiovascular disease as discussed by Muñana (1995). T1DM patients depend on a regular regiment of insulin injections and follow strict dietary restrictions in order to maintain their glucose within normal ranges. However, this is not an effective means of regulation as it depends on manual intervention and can cause fluctuations.

In recent years, the *Artificial Pancreas (AP) system* has been in development to serve as a viable alternative to the natural pancreas. The AP is a *safety-critical* control system that is made up of the following key components: (i) a continuous glucose monitor (CGM) which measures the blood glucose levels fairly continuously, (ii) an insulin pump that can deliver insulin subcutaneously in small doses and (iii) a control algorithm that computes the amount of insulin to be delivered by the pump. The control algorithm plays a major role in the AP system as accurate computation of insulin delivery can prevent hypoglycemia and hyperglycemia. There are several control techniques that have been implemented and tested for AP systems such as the standard Proportional Integral Derivative (PID) control, linear quadratic regulators (LQR), fuzzy logic controllers, sliding mode control, etc. More recently, *Model Predictive Control* (MPC) has proven to be a good candidate for this safety-critical system because of (i) its predictive nature which allows it to take action beforehand (ii) the constraint handling capabilities while (iii) utilizing minimal control effort. The major drawback of MPC is the computational requirements as it uses numerically intensive optimization techniques to find the optimal control. Therefore, a sufficiently powerful embedded device is required for an AP system that has an MPC technique at the center. The prime candidate for such a device is the modern-day smartphone because of its pervasiveness. In terms of processing power and memory capacities, they are comparable to laptops. This idea of developing AP systems around smartphones has been done in previous studies. The inter-operable artificial pancreas system (iAPS) Deshpande et al. (2019) from Harvard and the Diabetic Assistant (DiAs) Keith-Hynes et al. (2013) from the University of Virginia are examples of smartphone-based AP systems. The OpenAPS project is an open-source platform that is centered on Android smartphones which allow developers to modify and implement control techniques on their platform. The Android operating system is a suitable mobile platform to develop a fully functional AP system because of the availability of vast resources and computational toolboxes. Since there is significant research being done in smartphone-based AP systems, a comprehensive com-

[1] E-mails: abishekchandrasekhar92@gmail.com; padhi@iisc.ac.in

10.1016/j.ifacol.2022.04.092

parison of the efficiency of various optimization solvers will prove to be of use in future AP developmental activities.

A fully functional MPC-based AP system was developed by the authors using commercially available CGM sensors and insulin pumps. It is to be noted that any model-based control algorithm such as MPC requires state estimation to obtain the values of the states. In addition to the control algorithm, the authors have also included state estimation and additional safety logic for the AP system, but it will not be covered in the context of this paper. The major focus of this paper is the implementation of the Model Predictive Control (MPC) algorithm in the Android app. Two different approaches for implementing an MPC-based AP system are presented. In the first approach, the algorithm was developed using Simulink's Android support toolbox and an Android application was generated. Optimization solver from MATLAB's Optimization Toolbox (1997) were integrated as a part of the app. In this approach, there were restrictions on the types of solvers that could be implemented on an embedded device. The second approach was to implement the algorithm in a Python environment (see:Van Rossum and Drake Jr (1995)) and integrate the python code-base as a part of the Android app. Python was chosen as it has an established library of mathematical tools and several optimization packages. However, even though there are several optimization solvers available, not all of these solutions can be directly implemented on embedded platforms. A number of optimization packages (both in the MATLAB approach and Python approach) were implemented and evaluated on smartphones to test for compatibility. A comparison was made between all the compatible optimization solvers by running the MPC on an Android smartphone. A similar comparison of optimization solvers was done by Wendorff et al. (2016), but for the purpose of aircraft design. In the next section, the control algorithm development is presented, followed by the implementation and simulation using synthetic patient data.

2. CONTROL SYNTHESIS

As discussed in Section 1, the Model Predictive Control (MPC) is the control algorithm chosen for this AP system. MPC requires a mathematical model of the system (in this case a model describing the glucose-insulin interaction in T1DM patients).

2.1 System Model

The glucose-insulin dynamics extensively studied over the past few years. The model chosen has three sections: (i) A meal model developed by Dalla Man et al. (2007) that describes the glucose absorption process of a carbohydrate meal (ii) The minimal model developed by Bergman (1989) that describes the glucose-insulin interaction in the blood (iii) the subcutaneous insulin kinetics model developed by Schiavon et al. (2017) that describe how exogenous insulin finally appears in the blood (given by Eq.(3)). The equations of the augmented final model are shown below:

$$\begin{aligned} \dot{q}_{sto_1}(t) &= -k_{21} q_{sto_1}(t) + D\delta(t) \\ \dot{q}_{sto_2}(t) &= -k_{empty}\left(q_{sto}\right) q_{sto_2}(t) + k_{21} q_{sto_1}(t) \\ \dot{q}_{gut}(t) &= -k_{abs} q_{gut}(t) + k_{empty}\left(q_{sto}\right) q_{sto_2}(t) \end{aligned} \tag{1}$$

$$\begin{aligned} \dot{G}(t) &= -p_1 G(t) - X(t)G(t) + p_1 G_b + \frac{f k_{abs} q_{gut}(t)}{V_g} \\ \dot{X}(t) &= -p_2 X(t) + p_3 \left(\frac{I_p(t)}{V_I} - I_b\right) \end{aligned} \tag{2}$$

$$\begin{aligned} \dot{I}_{sc_1}(t) &= -\left(k_{a_1} + k_d\right) I_{sc_1} + u(t-\tau) \\ \dot{I}_{sc_2}(t) &= -k_{a_2} I_{sc_2} + k_d I_{sc_1} \\ \dot{I}_p(t) &= -k_e I_p(t) + k_{a_1} I_{sc_1} + k_{a_2} I_{sc_2} \end{aligned} \tag{3}$$

In the meal model shown in Eq.(1), q_{sto_1}, q_{sto_2} represent the appearance of the carbohydrate food in the stomach in the solid phase and liquid phase respectively. q_gut represents the carbohydrate absorption in the intestines. In the glucoregulatory model (given by Eq.(2)), G represents the glucose in the blood and X represents the remote insulin that interacts with glucose. The subcutaneous kinetics is given in Eq.(3) where the input u is the rate of insulin that is infused subcutaneously. This insulin appears in the first compartment after a time delay τ. I_{sc_1} and I_{sc_2} represent the insulin in *non-monomeric* and *monomeric* form respectively and I_p is the insulin in the plasma. The details of the parameters are not described here for brevity and can be found in the references mentioned above. This augmented model can be represented in state-space form as $\dot{\mathbf{X}} = f(\mathbf{X}, \mathbf{U})$ where $\mathbf{X} \in \mathbb{R}^8$ is a vector of the *state variables* defined as $\mathbf{X} \triangleq [q_{sto_1}, q_{sto_2}, q_{gut}, G, X, I_{sc_1}, I_{sc_2}, I_p]^T$.1

2.2 MPC Formulation

The MPC formulation utilizes the discretized version of the system dynamics model that is described in the previous section2.1. The standard Euler method is used for discretization. The objective is to minimize the deviations of glucose G_k and insulin I_k from their basal levels by finding the set optimal insulin infusion rate U_k while respecting system dynamics and bounds on the glucose state. The input is also constrained based on the actuator limitations (in this case, the limitations of the insulin pump). The cost function is formulated as follows

$$\begin{aligned} \min_{\mathbf{U}} J = &\sum_{k=1}^{N} \left(Q_1(G_k)(G_k - G_{ref})^2 + Q_2(I_k - I_{ref})^2\right) \\ &+ \sum_{j=1}^{N_c} R(U_j - U_{ref})^2 \end{aligned} \tag{4}$$

In Eq.(4) the reference values for the terms G_{ref}, I_{ref} are the steady state/basal values which vary subject to patient. U_{ref} is the basal delivery rate for the patient. The control U, which is the optimization variable is updated once every 5 minutes. Q_1, Q_2 and R are the weights associated with G, I and U. The weight Q_1 is state-dependent and is formulated such that cost function is not penalized if G is lower than G_{ref}. Note that N and N_c are the prediction horizon and the control horizon respectively. The performance index is subject to the following constraints which are given below in Eq.(**??**):

$$
\begin{aligned}
&\textit{subject to}: \\
&\mathbf{X_{k+1}} = f(\mathbf{X}, \mathbf{U}) \\
&G_{l_b} \leq G_k \leq G_{u_b} \\
&U_{l_b} \leq U_k \leq U_{u_b} \\
&U_{sum} \triangleq \sum_{k=0}^{N_c} U_k \leq U_t
\end{aligned} \tag{5}
$$

The details and the justifications for each of the constraints are described in detail below:

- The glucose-insulin dynamic behavior that is given in Eqs. (1) - (3)
- The blood glucose G should always remain bounded between the minimum and maximum values to avoid hypoglycemia G_{l_b} and hyperglycemia G_{u_b}. These bounds ensure the safety-critical functioning of the AP system.
- The control input is bounded based on the maximum delivery rate constraint of the insulin pump U_{u_b}.
- The total amount of insulin delivered to the patient is limited by U_t to prevent insulin overdose which may lead to hypoglycemia.

The sampling time for numerical integration of the plant model was selected to be *1 minute* as the dynamics of this system are relatively slow. Therefore, an optimal control problem is converted to a constrained optimization problem using the *Transcription method*. The process of solving an optimization problem where constraints are nonlinear is called *nonlinear programming* (NLP). It should be noted that the proposed problem is nonlinear because of the glucose-insulin dynamic model. There are several techniques that have been developed for solving nonlinear constrained optimization problems such as Sequential Quadratic Programming (SQP), Interior Point method, and Trust-region method. The next section focuses on the implementation of the MPC algorithm.

3. ANDROID IMPLEMENTATION OF NLP SOLVERS FOR MPC

The challenge in developing an MPC-based AP system is the implementation of the system on a portable embedded computer. As mentioned in th11, the AP system proposed here has been developed centered around an Android smartphone (which is the embedded computer in this case). The first step is to develop a framework that can allow the MPC to be solved on a smartphone.

3.1 Framework Design for evaluation

In order to evaluate the various NLP solvers, the implementation of MPC is done in two different frameworks. The approaches were to: (i) Develop the Android app using Simulink's Android support package (ii) Develop the app in the Python environment and integrate the Python library with the Android App.

Simulink/MATLAB Approach: The MPC formulation detailed in Section 2.2 was implemented in the Simulink environment. The *fmincon* function (see: Optimization-Toolbox (2021a)) from the optimization toolbox was added to solve the NLP problem. The solvers that are available in the fmincon function are : Sequential Quadratic Programming, Interior-Point method, Trust-Region method. The block-sets from the Android Support Toolbox were incorporated into the Simulink model to make the algorithm compatible with the Android environment. An Android application was then generated from the Simulink model by configuring the model using the Hardware and Deployment toolbox. The generated app was then imported into the Android Studio platform in order to add a user-interface make modifications to enable additional components like communication protocols and a database management system. The details of the app generation and modification process are not detailed here as it is not the focus of this paper. The basic framework of the android app generated from MATLAB is shown below in Fig. 1.

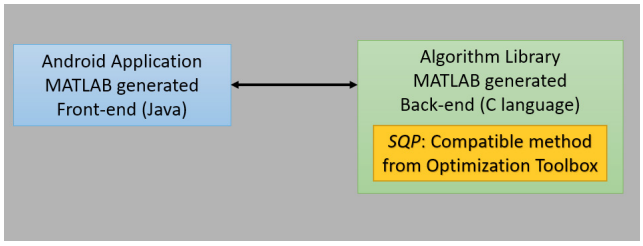

Fig. 1. Front-end and Back-end of MATLAB generated application

The front-end user interface that displays the output is in Java and the back-end MPC algorithm is generated in C language. As the application is auto-generated, there is no need for adding a manual interface between the front-end and the back-end.

Python Approach: The same formulation given in Section 2.2 was implemented in the Python environment. Python has a large number of optimization libraries such as Casadi, Scipy, Pymoo, Pagmoo etc. Each optimization library has a similar set of NLP solvers just as the MATLAB optimization toolboxes. The python MPC library that was developed for the AP problem was integrated to the android app via the Chauqopy toolbox. The basic framework for the Python based approach is shown below in Fig. 2.

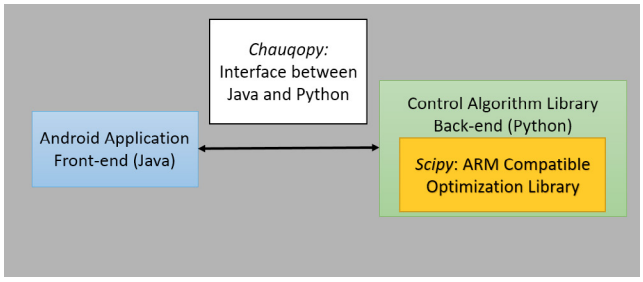

Fig. 2. Front-end and Back-end of Python based application

The second major step is to find and evaluate NLP solvers that are compatible for this Android based AP system.

3.2 Compatible Optimization Solvers

The authors found that most optimization solvers were developed for **x**86 based processors that are commonly found

in laptops and desktops. Most portable computing devices such as smartphones and tablets, however, use **ARM** based processors. A number of optimization libraries were evaluated and a list of compatible toolboxes/libraries was evaluated for mobile implementation. In addition to this, a number of optimization tools required code generation and were not directly compatible with the Android environment. A summary of various optimization solvers can be found in Table 1.

Table 1. Survey of Optimization Solvers

Environment	Toolbox/Function	Comment
MATLAB	Optimization Tool	Generates Android App
MATLAB/Python	Casadi	Generates C++ code
Python	qpsolvers	No support for NLP
Python	Scipy	Directly Implementable
Python	Pymoo	Not compatible
Python	lpsolvers	No support for NLP
Python	GEKKO	Directly Implementable

In MATLAB's Optimization toolbox, the *fmincon* function includes popular algorithms to solve NLP problems such as interior-point, trust-region-reflective and SQP. However, it was found that only the SQP method was compatible for implementation on embedded systems/Android phones. The supporting documentation for this can be found at Optimization-Toolbox (2021b). The advantage of using the direct app generation from Simulink's Android support toolbox is that the generated code for the optimization library is automatically linked to the front-end of the Android App. In the case of Python, as mentioned, there are a variety of python optimization libraries that are available such as Casadi, Scipy, Pymoo etc. Casadi is a popular tool for solving optimization problems and optimal control problems. However the Casadi library cannot be implemented directly in Python environment and requires code-generation (in C++) which is then to be integrated to the Android App manual. The Casadi library can also be implemented in MATLAB, but the same issue remains where a C++ code is to be generated. Therefore, Casadi's library was not selected as a candidate in both approaches as it requires manual modifications to interface it to the android app for integration. In the case of other popular solvers such as qpsolvers, they lacked support for adding nonlinear constraints as a part of the optimization problem. Two compatible toolboxes were the Scipy toolbox and the Gekko toolbox in Python. However, the Gekko toolbox was not used for the final evaluation, as the Scipy toolbox served the need. The final libraries and algorithms chosen for evaluation are:

Table 2. Final Compatible Optimization Solvers

Environment	Toolbox/Function	Solver
MATLAB	fmincon	SQP
Python	Scipy	SLSQP, Trust-Region

Note: The abbreviations for solvers and libraries are as follows: SQP: Sequential Quadratic Programming, SLSQP: Sequential Least Squares Programming A brief introduction into the chosen methods are given below:

Sequential Quadratic Programming (SQP): This is one of the most commonly used methods for solving constrained nonlinear optimization problems. It is the only NLP solver from the Optimization toolbox in MATLAB that was compatible with Android phones. It is an iterative method and at each iteration, a Quadratic Programming problem is solved by making approximations of Lagrangian (see: Fletcher (2013)). It is to be noted that the initialization is to be done such that the point is feasible.

Sequential Least-Squares Quadratic Programming (SLSQP): This is a type of SQP method that was developed by Kraft et al. (1988). In SLSQP, the NLP is considered to be a sequence of constrained-least squares problems. Just as in the SQP method mentioned above, approximations of the objective functions are done using the Broyden–Fletcher–Goldfarb–Shanno (BFGS) update (see: Head and Zerner (1985)).

Trust-Region Constrained Algorithm: This method is a part of the NLP solvers available in Scipy. It uses the *Interior-point algorithm* (see: Byrd et al. (1999)) to handle inequality constraints where a "trust" region is constructed around the current solution and the objective function is approximated to a simpler function.

4. SIMULATION

The two frameworks that were proposed were tested using synthetic patient data. The results obtained were verified and compared for each of the solvers.

4.1 Test Cases

The MPC problem for the AP system described in Section 2 was simulated for the following conditions:

- Prediction horizon N: 120 minutes
- Control horizon N_c: 90 minutes
- Glucose upper bound G_{ub}: 240 mg/dL
- Glucose lower bound G_{lb}: 70 mg/dL
- Control upper bound U_{ub}: 1.5 Units/min
- Control lower bound U_{lb}: 0 Units/min
- Integral Constraint U_{total}: 25 Units

The robustness of the model predictive control has also been tested for parameter uncertainty, initial condition uncertainties and with state-estimation in the loop. The details of the tests are omitted here for brevity. A simulation of the algorithm with parameter uncertainty (±20%) and initial condition uncertainty is shown below in Fig. 3 where the glucose values are plotted for different uncertainties. It can be seen that the values remain within the bound for almost all the 30 different variation tests.

4.2 Implementation Results

The AP algorithms were run on the Samsung Galaxy A31 android phone which has an 8 core processor (2x2.0 GHz, 6x1.7 GHz) and 6 gigabytes of RAM. The execution time of the various solvers are compared for 10 runs on the Android phone. The Python and the MATLAB-based frameworks are tested first and the maximum number of iterations for the solver is set to 200. It should be noted

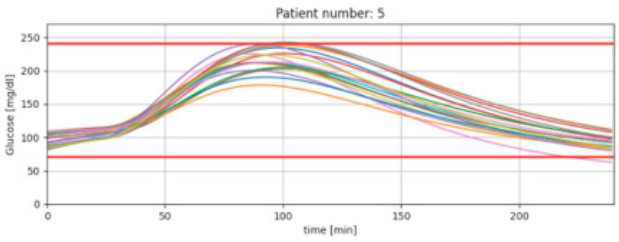

Fig. 3. Closed Loop Glucose (uncertainty test)

that the same exact insulin command ($0.6U/min$) was obtained from each run.

Note: The proposed approaches were also tested on smartphones with different memory and performance capacities to check for consistency and accuracy of the results. (Motorola X4 with 4 gigabytes of RAM and the Samsung Galaxy S10+ 8 gigabytes of RAM) . However, the comparison of execution times on these different phones is not reported here as this work mainly focuses on the software architecture performance on one phone.

Iterations to Solution: The number of iterations taken by each NLP solver to converge to the solution was recorded. It is to be noted that the in initial conditions were identical in every test case. The results for one particular case is shown in Table 3. These results are for a case where a feasible solution was found.

Table 3. Iterations for different solvers

NLP Solver	Iterations
Trust-region-Python	83
SLSQP-Python	151
SQP-MATLAB	65

The NLP solvers that converges to the solution in the least number of iterations was the SQP method and the solver that took the most number of iterations was the SLSQP method from the Python framework.

Execution Time: The time taken by various solvers to converge to a solution is shown in Fig. 4. It can be seen that

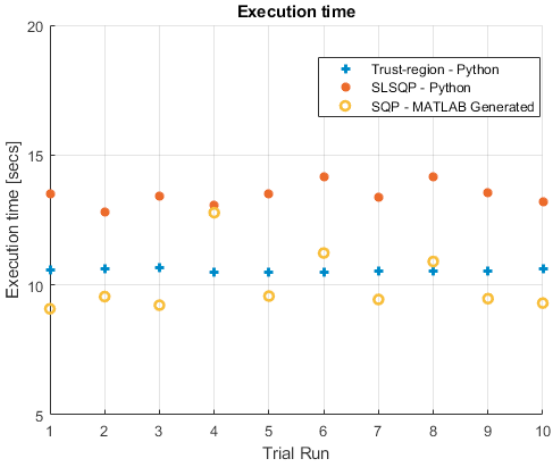

Fig. 4. Time taken by Optimization Solvers to converge to Minima

there is a negligible difference in execution times of the three NLP solvers that were tested. The SLSQP method from the Python framework takes the most time to execute (average of *13.5* seconds) whereas both the trust-region constrained method from the Python framework and the SQP method from the MATLAB framework completed the optimization at approximately *10.5* seconds on average. It should be noted that a convergence time of *10-13 seconds* is sufficiently fast for this system as the control input is only updated once every *5 minutes*.

Ease of implementation: Based on the above results, it can be seen that both the Scipy based python approach as well as the app generation from Simulink approach offer negligible difference in performance. Therefore the next logical comparison would be to compare the ease of development and implementation.

(i) The MATLAB/Simulink Approach: Although generating an Android app from MATLAB/Simulink seems straight forward, the generated application cannot be used directly interface with the physical components of the system. In-order to develop a full- fledged AP system with a user-interface and hardware connectivity (such as sensors and insulin-pumps), there are significant modifications needed for the MATLAB approach. This process of generating an application and the modifications required are shown by our previous work in Chandrasekhar et al..

(ii) The Python Approach: Compared to the MATLAB/Simulink method of developing an AP system, the Scipy based Python approach is fairly straight forward interms of implementation as there is no code generation involved and the further improvements/features can be easily added.

Advantages of proposed architectures It should also be noted that both these approaches allow for rapid development and implementation of AP systems when compared to traditional architectures built on C/C++ (which offer faster computation times). The modular nature of the proposed architectures also easy implementations of algorithm improvements in the future.

4.3 Real-world Testing

The two approaches reported in this work have also been successfully tested on multiple Type-1 diabetic patients in a clinical setting. The results are omitted hear as they are out of the scope of this work. The clinical trial results will be shown in future works of the authors.

5. CONCLUSION

In this work, two different ways of implementing MPC for a smartphone-based Artificial Pancreas system are presented. A MATLAB-based approach, where an Android app can be generated from a Simulink model, and a Python-based approach where python libraries are interfaced with the Java application. A survey of various optimization tools and solvers was done and those that were easily implementable and compatible on Android smartphones were tabulated. It was found that the Sequential Quadratic Programming method in MATLAB's optimization toolbox was the only Android compatible nonlinear optimization solver. In the Python framework, it was

shown that the Trust-region method and the Sequential Least Squares Quadratic Programming method were both compatible with Android phones. These frameworks were tested on an Android phone and it was seen that the SQP method using the MATLAB framework converges to a solution with fewer iterations whereas the SLSQP method using the Python framework takes the most number of iterations. A comparison of the execution times of various solvers was also presented, and it was shown that there was a negligible difference in execution times. In a slow control system such as this, the execution time difference of a few seconds will not play any role in performance. It can be concluded that the Python based Scipy toolbox along with the Trust-region constrained method is the ideal candidate for such an application. Using this development method, rapid prototyping can be done for continuous improvement. The work presented in this paper can be used for the development of MPC on smartphone-based AP systems. This work can also be used to implement MPC on other ARM processor-based systems such as a Raspberry Pi.

REFERENCES

Bergman, R.N. (1989). Toward physiological understanding of glucose tolerance: minimal-model approach. *Diabetes*, 38(12), 1512–1527.

Byrd, R.H., Hribar, M.E., and Nocedal, J. (1999). An interior point algorithm for large-scale nonlinear programming. *SIAM Journal on Optimization*, 9(4), 877–900.

Chandrasekhar, A., Saini, D., and Padhi, R. (????). An android phone based artificial pancreas system for type-1 diabetic patients. In *2021 IEEE 9th Region 10 Humanitarian Technology Conference (R10-HTC)*, 1–6. IEEE.

Dalla Man, C., Rizza, R.A., and Cobelli, C. (2007). Meal simulation model of the glucose-insulin system. *IEEE Transactions on biomedical engineering*, 54(10), 1740–1749.

Deshpande, S., Pinsker, J.E., Zavitsanou, S., Shi, D., Tompot, R., Church, M.M., Andre, C., Doyle III, F.J., and Dassau, E. (2019). Design and clinical evaluation of the interoperable artificial pancreas system (iaps) smartphone app: interoperable components with modular design for progressive artificial pancreas research and development. *Diabetes technology & therapeutics*, 21(1), 35–43.

Fletcher, R. (2013). *Practical methods of optimization*. John Wiley & Sons.

Head, J.D. and Zerner, M.C. (1985). A broyden—fletcher—goldfarb—shanno optimization procedure for molecular geometries. *Chemical physics letters*, 122(3), 264–270.

Keith-Hynes, P., Guerlain, S., Mize, B., Hughes-Karvetski, C., Khan, M., McElwee-Malloy, M., and Kovatchev, B.P. (2013). Dias user interface: a patient-centric interface for mobile artificial pancreas systems.

Kraft, D. et al. (1988). A software package for sequential quadratic programming.

Muñana, K.R. (1995). Long-term complications of diabetes mellitus, part i: Retinopathy, nephropathy, neuropathy. *Veterinary Clinics of North America: Small Animal Practice*, 25(3), 715–730.

Optimization-Toolbox (2021a). Mathworks optimization. URL `https://in.mathworks.com/help/optim/ug/fmincon.html`.

Optimization-Toolbox, S. (2021b). Compatible mathworks optimization. URL `https://in.mathworks.com/help/optim/ug/code-generation-in-fmincon.html`.

Schiavon, M., Dalla Man, C., and Cobelli, C. (2017). Modeling subcutaneous absorption of fast-acting insulin in type 1 diabetes. *IEEE Transactions on Biomedical Engineering*, 65(9), 2079–2086.

Toolbox, M.O. (1997). User's guide. *The MathWorks, Inc.–2019.–3788* .

Van Rossum, G. and Drake Jr, F.L. (1995). *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam.

Wendorff, A., Botero, E., and Alonso, J.J. (2016). Comparing different off-the-shelf optimizers' performance in conceptual aircraft design. In *17th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, 3362.