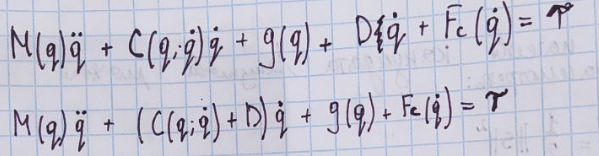
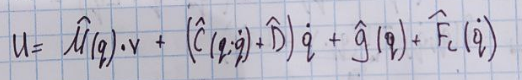
**Экзаменационная работа**

**Робастный контроллер**

В случае управления манипулятором чаще всего неизвестны точные параметры системы. В таком случае можно применить робастное управление. Система, включающая демпфирование и кулоновское трение:

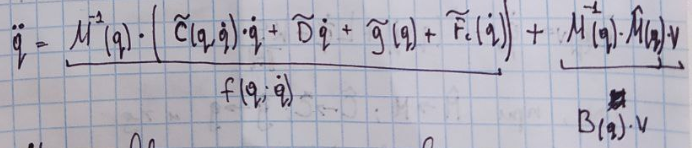


Систему можно попытаться линеаризовать в управлении при помощи оценок параметров:

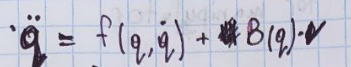


Однако, правильнее посчитать, что демпфирование и кулоновское трение полностью неизвестны, так как не всегда есть возможность их оценить или измерить.

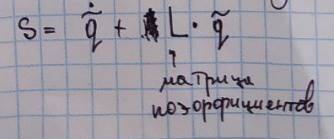
При подстановке такого управления в исходную систему мы получаем:



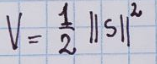
,где параметры с волной – разность оценки и истинного значения.

Произведя указанную замену, мы получаем уравнение вида: 

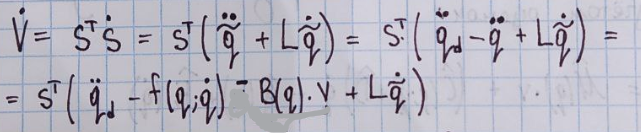
Далее можно ввести скользящую поверхность. Так как в нашем случае система второго порядка, уравнение скользящей поверхности будет выглядеть следующим образом:



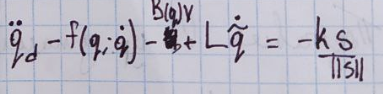
Теперь для определения управления необходимо воспользоваться сходимостью по Ляпунову. В качестве кандидата можно выбрать:

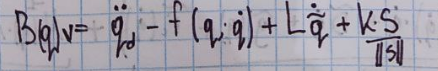


Производная по времени:



Чтобы производная была всегда меньше 0:

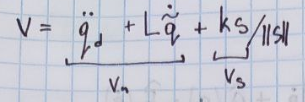




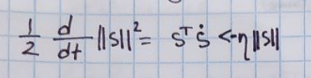
Если же оценки достаточно близки, то:

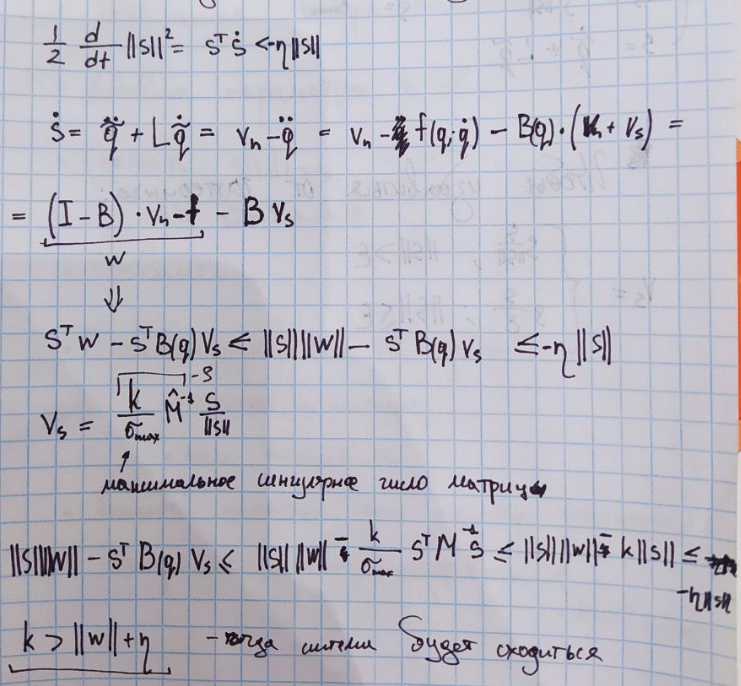


В таком случае:

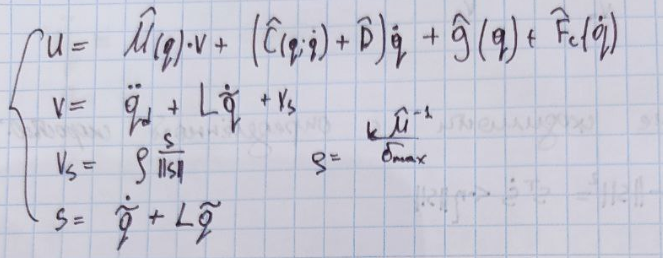


Из условия сходимости с определенной скоростью можно вывести условие, налагаемое на коэффициент:



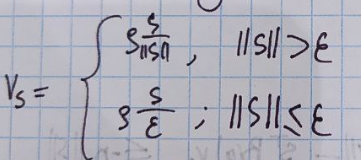


Конечное управление представляет собой:



Опять же, можно посчитать, что трение нам не известно, и не включать его в управление.

Главным недостатком такого управления является чаттеринг, который возникает из-за быстрой смены знака . Чтобы избавиться от него можно вычислять по системе:



Тогда при правильно подобранном , система не будет колебаться.

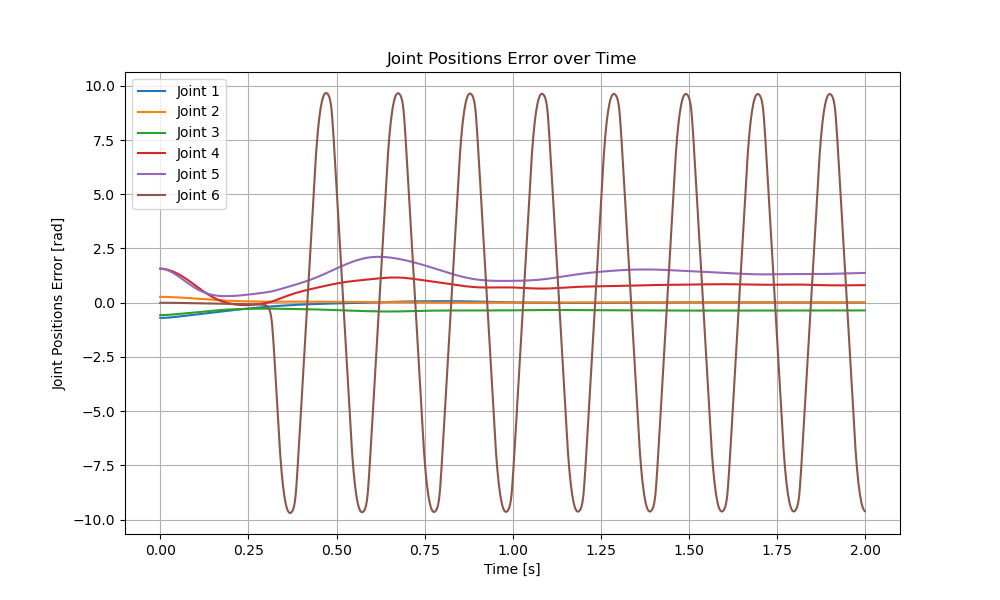
**Сравнение робастного контроллера с ПД**

В python с использованием библиотеки Pinocchio и симулятора MuJoCo было написано робастное управление манипулятором, также был написан классический ПД контроллер.

В симуляторе было задано трение и масса эндефектора.

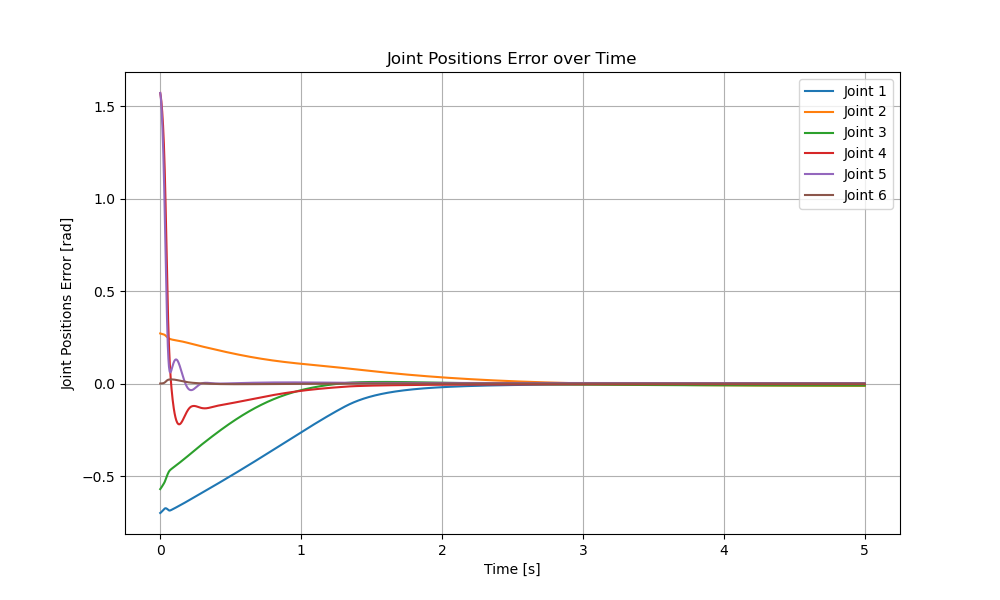
Исходный код, видео симуляции и графики представлены в репозитории.

Из результатов видно, что ПД контроллера совсем не справляется с управлением системой:



Ошибка не сходится к 0, а последний джоинт вообще совершает колебания с большой частотой.

При этом в случае робастного управления получается следующий результат:



Видно, что ошибка управления сходится в ноль.

Параметры робастного контроллера подбирались вручную, согласно выведенному неравенству.

Параметр определялся таким образом, чтобы исключить чаттеринг.

В моем случае пороговое значение, про котором пропадал чаттеринг - = 55.

Графики ошибки и подаваемого управления для разных :

|  |  |
| --- | --- |
| = 30 | |
| C:\Users\Valiu\Desktop\Mag1.1\Control\Control_Exam\robust\plots_for_dif_eps\eps30\Exam_robust_err.png | C:\Users\Valiu\Desktop\Mag1.1\Control\Control_Exam\robust\plots_for_dif_eps\eps30\Exam_robust_tor.png |
| = 50 | |
| C:\Users\Valiu\Desktop\Mag1.1\Control\Control_Exam\robust\plots_for_dif_eps\eps50\Exam_robust_err.png | C:\Users\Valiu\Desktop\Mag1.1\Control\Control_Exam\robust\plots_for_dif_eps\eps50\Exam_robust_tor.png |
| = 55 | |
| C:\Users\Valiu\Desktop\Mag1.1\Control\Control_Exam\robust\plots_for_dif_eps\eps55\Exam_robust_err.png | C:\Users\Valiu\Desktop\Mag1.1\Control\Control_Exam\robust\plots_for_dif_eps\eps55\Exam_robust_tor.png |
| = 60 | |
| C:\Users\Valiu\Desktop\Mag1.1\Control\Control_Exam\robust\plots_for_dif_eps\eps60\Exam_robust_err.png | C:\Users\Valiu\Desktop\Mag1.1\Control\Control_Exam\robust\plots_for_dif_eps\eps60\Exam_robust_tor.png |
| = 100 | |
| C:\Users\Valiu\Desktop\Mag1.1\Control\Control_Exam\robust\plots_for_dif_eps\eps100\Exam_robust_err.png | C:\Users\Valiu\Desktop\Mag1.1\Control\Control_Exam\robust\plots_for_dif_eps\eps100\Exam_robust_tor.png |

Вывод: При правильно подобранных параметрах робастный контроллер способен управлять системой с неизвестными точно параметрами, что выгодно отличает его от обычного ПД контроллера.