Model: Default (GPT-3.5)

Write C++ code to implement model-predictive control for an HVAC system.

Implementing model-predictive control for an HVAC system requires a number of components, including a mathematical model of the system, an optimization problem to solve for control inputs, and a control loop to apply the optimized inputs to the system. Here's an example implementation in C++:

c++ Copy code

```c++
#include <iostream>
#include <vector>
#include <Eigen/Dense>
#include <qpOASES.hpp>

using namespace Eigen;
using namespace std;
using namespace qpOASES;

// Define the mathematical model of the HVAC system.
MatrixXd A(2, 2); // State transition matrix
MatrixXd B(2, 1); // Control input matrix
VectorXd x0(2); // Initial state
VectorXd x(2); // Current state
VectorXd y(1); // Output (temperature)

void initModel() {
  // Define system matrices
  A << 0.95, 0.05, 0, 1;
  B << 0.1, 0;
  x0 << 20, 0;
  x = x0;
}

// Define the cost function and constraints for the optimization problem.
const int N = 10; // Prediction horizon
const int nu = 1; // Control input dimension
const int nx = 2; // State dimension

MatrixXd Q(nx, nx); // State penalty matrix
MatrixXd R(nu, nu); // Control penalty matrix
VectorXd x_ref(nx); // Reference state
double u_min = -1.0; // Minimum control input
double u_max = 1.0; // Maximum control input

void initController() {
  // Define cost function parameters
  Q << 1, 0, 0, 0.1;
  R << 0.1;
  x_ref << 23, 0;
}

// Solve the optimization problem using qpOASES.
real_t H[(N+1)*nu][(N+1)*nu];
real_t g[(N+1)*nu];
real_t A_cons[N*nx][(N+1)*nu];
real_t lb_cons[N*nx];
real_t ub_cons[N*nx];
real_t lb_u[N];
real_t ub_u[N];
real_t x_opt[(N+1)*nu];

void solveQP() {
  // Construct the Hessian matrix and gradient vector.
  for (int i = 0; i <= N; i++) {
    for (int j = 0; j <= N; j++) {
      if (i == j) {
        for (int k = 0; k < nu; k++) {
          H[i*nu+k][j*nu+k] = R(k, k);
        }
      } else {
        for (int k = 0; k < nu; k++) {
          H[i*nu+k][j*nu+k] = 0;
        }
      }
    }
  }
  for (int i = 0; i <= N; i++) {
    for (int j = 0; j < nu; j++) {
      if (i == 0) {
        g[i*nu+j] = (B.transpose() * Q * (x_ref - x)).coeff(0, j);
      } else if (i <= N) {
        g[i*nu+j] = 0;
      }
    }
  }

  // Construct the inequality constraints.
  for (int i = 0; i < N; i++) {
    for (int j = 0; j < nu; j++) {
      for (int k = 0; k < nx; k++) {
        A_cons[i*nx+k][i*nu+j] = (A.pow(i
```

continue

css Copy code

```css
      ) * B).coeff(k, j);
    }
    for (int k = 0; k < nx; k++) {
      A_cons[i*nx+k][(i+1)*nu+j] = (A.pow(i+1) * B).coeff(k, j);
    }
    lb_cons[i*nx+j] = -numeric_limits<double>::infinity();
    ub_cons[i*nx+j] = numeric_limits<double>::infinity();
  }
```

}

// Construct the control input constraints.
for (int i = 0; i < N; i++) {
lb_u[i] = u_min;
ub_u[i] = u_max;
}

// Solve the QP.
real_t* H_ptr = &H[0][0];
real_t* g_ptr = &g[0];
real_t* A_ptr = &A_cons[0][0];
real_t* lb_ptr = &lb_cons[0];
real_t* ub_ptr = &ub_cons[0];
real_t* lb_u_ptr = &lb_u[0];
real_t* ub_u_ptr = &ub_u[0];
real_t* x_opt_ptr = &x_opt[0];
QProblem qp((N+1)nu, Nnx);
Options options;
options.printLevel = PL_NONE;
qp.setOptions(options);
qp.init(H_ptr, g_ptr, A_ptr, lb_ptr, ub_ptr, lb_u_ptr, ub_u_ptr, N, NULL);
qp.getPrimalSolution(x_opt_ptr);
}

// Apply the optimized control input to the system and update the state.
void applyControlInput() {
double u = x_opt[0];
u = max(u, u_min);
u = min(u, u_max);
x = A * x + B * u;
}

int main() {
initModel();
initController();

// Run the control loop.
for (int i = 0; i < 100; i++) {
solveQP();
applyControlInput();
y = x(0); // Output temperature
cout << "Time: " << i << ", Control input: " << x_opt[0] << ", Output: " << y(0) <<
endl;
}

return 0;
}

css Copy code

```css
ou can modify this to run for any length of time or until some stopping crite
```