The culmination of this project was implemented in a new force control behavior titled ‘ForceControlBehaviourAdaptive’ in the ‘ForceControlBehaviours’ Class. This adaptive mode is accessible to the user through the Galen Robot GUI and is titled ‘FTVELOCITY_ADAPTIVE’. This mode of operation can be selected instead of the customary ‘FTVELOCTY’ option.

```
ForceControlBehavior::ForceControlBehavior(robotTask *robot, std::shared_ptr<robotModel> model, std::shared_ptr<IGalenLogger> logger)
BehaviorBase("ForceControlBehavior", model, logger),
StateTable(5000, "StateTable"),
RobotTask(robot),
m_pLogger(logger)
{
  AddMode(Mode::BehaviorManagerStrings::LinearForceControl);  //mode 1.
  AddMode(Mode::BehaviorManagerStrings::LinearForceControl_SIM); //mode 2.
  AddMode(Mode::BehaviorManagerStrings::LinearForceControl_Adaptive); //mode 3.
```

To implement the corrected control loop timing, two new functions were added to the ‘RobotTask’ class. The function ‘SavePrevJointVelSet’ can be used to store current joint velocity goals in memory after they have been calculated by the optimizer. The function ‘GetPrevJointVelSet’ can be used to retrieve the joint velocity goal calculated in the previous timestep.

```
//store the velocity goal after optimizer calculates it (CIS2 project Vishnu Kolal)
void robotTask::SavePrevJointVelSet(const mtsDoubleVec& prevVelSet)
{
    prevJointVelSet = prevVelSet;
}

//retrieve the velocity goal to send to galil (CIS2 project Vishnu Kolal)
mtsDoubleVec robotTask::GetPrevJointVelSet() const
{
    mtsDoubleVec retVec;
    retVec.SetSize(5);
    retVec.Zeros();

    retVec = prevJointVelSet;

    return retVec;
}
```

Implementation of adaptive gains:

```
void robotTask::UpdateJointGains(void)
{
    if(PedalLR[0] > 0.0)
    {
        // p * mf * (maxVel * dtOpt)/maxFT
        gains[0] = PedalLR[0] * SliderGains_R * scaleFactor[0];
        gains[1] = PedalLR[0] * SliderGains_R * scaleFactor[1];
        gains[2] = PedalLR[0] * SliderGains_T * scaleFactor[2];  //invert to correspond to corrected FT
    }
    else
    {
        gains[0] = gains[1] = gains[2] = 0;
    }

    mtsDouble nextAdaptiveFactor;
    if(PedalLR[0] > 0.9 && forceAtHandle.XYZ().Norm() > adaStartForce)
    {
            if(adaElapsedCount * StateTable.Period < adaStartPeriod)
            {
                adaElapsedCount++;
            }
            else if(adaElapsedCount * StateTable.Period >= adaStartPeriod)
            {
                nextAdaptiveFactor = adaptiveFactor + ( (adaptiveMax - adaptiveMin) / ( adaRampUpPeriod/StateTable.Period ) );                          //ramp up
                if(nextAdaptiveFactor * scaleFactor[2] <= adaMaxGain && nextAdaptiveFactor <= adaptiveMax) { adaptiveFactor = nextAdaptiveFactor;}
            }
    }
    else
    {
        if(adaptiveFactor > adaptiveMin)
        {
            adaptiveFactor = adaptiveFactor - ( (adaptiveMax - adaptiveMin) / ( adaRampDownPeriod/StateTable.Period ) );                          //ramp down
        }
        else if(adaptiveFactor <= adaptiveMin)
        {
            adaptiveFactor = adaptiveMin;
        }
        adaElapsedCount = 0;
    }
```

The adaptive gains variable ‘adaptiveFactor’ was added to the ‘UpdateJointGains’ method in the class ‘RobotTask’. This variable is kept updated at all times but is only being used by the adaptive force control behavior.

The variables being used for the adaptive gains are initialized in the constructor for the ‘RobotTask’ class and can be edited here.

```
//variables for adaptive gains (CIS-2 project -Vishnu Kolal)
adaptiveFactor = 1;         //adaptive gain (should be 1)
adaptiveMin = 0.6;            //min adaptive gain
adaptiveMax = 5.8;          //max adaptive gain
adaElapsedCount = 0;        //counter for duration (should be 0)

adaStartForce = 4.20;       //Force above which duration starts (N)
adaStartPeriod = 2.80;     //duration after which adaptive gain kicks in (s)
adaRampUpPeriod = 2.1;      //duration for the adaptive gain to reach its max limit (s)
adaRampDownPeriod = 0.40;    //duration for the adaptive gain to come back to normal (s)
adaMaxGain = 15;            //max scalefactor*adaptiveFactor allowed in adaptive mode
adaTemp = 0;                //ignore
```

Implementation of sensor data filtering:

```
if(isRawFTDataOkay && isPostAlignmentFTDataOkay)
{
    //Applying Frame Transformations
    forceAtHandle = robotModel::TransformForceToOutputFrame(forceAtEE, toolHandleFrame);
    model->ApplyThresholdToFTVector(forceAtHandle, ForceThreshold.Data, ForceSensor::torque_threshold_Nm);
    for(size_t i=filterBuffer; i>0; i--)
    {
        forceAtHandleFilt[i] = forceAtHandleFilt[i-1];
    }
    forceAtHandleFilt[0] = forceAtHandle;
    mtsDouble forceWeight = 1;
    forceAtHandleSmooth.Zeros();
    for(size_t i=0; i<filterBuffer; i++)
    {
        forceWeight = forceWeight/2;
        for(size_t j=0; j<6; j++)
        {
            forceAtHandleSmooth[j] = forceAtHandleSmooth[j] + forceWeight*forceAtHandleFilt[i][j];
        }
    }
```

This algorithm was also implemented directly in the ‘UpdateFTData’ method of the ‘RobotTask’ class and is called in every iteration. The implementation of this sensor data filtering does not affect other modes of operation of the robot because the filtered data is stored in a separate variable called ‘forceAtHandleSmooth’ which is only being used by ‘ForceControlBehaviourAdaptive’. This filtered force is however, available to all methods.